

If less than the entire record is written, the remaining portion of the record is filled with blanks. The file position is always at the beginning of the file prior to execution of the I/O statement. Internal files permit only formatted, sequential I/O; and only the I/O statements READ and WRITE may specify an internal file.

Internal files provide a mechanism for using the formatting capabilities of the I/O system to convert values to and from their external character representations within the MS-FORTRAN internal memory structures. That is, reading a character variable converts the character values into numeric, logical, or character values and writing a character variable allows values to be converted into their (external) character representation.

The backslash edit descriptor (\) may not be used with internal files.

4.2.3 Units

A unit is a means of referring to a file. A unit specified in an I/O statement is either an external unit specifier or an internal unit specifier.

1. External unit specifier

An external unit specifier is either an integer expression (which evaluates to a nonnegative value) or the character * (which stands for the screen (for writing) and the keyboard (for reading)).

In most cases, an external unit specifier value is bound to a physical device (or files resident on the device) by name, using the OPEN statement. Once this binding of unit to system filename occurs, MS-FORTRAN I/O statements specify the unit number to refer to the associated external entity. Once the file is opened, the external unit specifier value is uniquely associated with a particular external entity until an explicit CLOSE operation occurs or until the program terminates.

The only exception to these binding rules is that the unit value zero is initially associated with the keyboard for reading and

the screen for writing and no explicit OPEN statement is necessary. The MS-FORTRAN file system interprets the character * as unit zero.

2. Internal unit specifier

An internal unit specifier is a character variable or character array element that directly specifies an internal file.

See Section 4.3.1, "Elements of I/O Statements," for a discussion of how these unit specifiers are used.

4.2.4 Commonly Used File Structures

Numerous combinations of file structures are possible in MS-FORTRAN. However, two kinds of files suffice for most applications:

1. * files
2. named, external, sequential, formatted files

* represents the keyboard and screen, that is, a sequential, formatted file, also known as unit zero. When reading from unit zero, you must enter an entire line; the normal operating system conventions for correcting typing mistakes apply.

An external file can be bound to a system name by any one of the following methods:

1. If the file is explicitly opened, the name can be specified in the OPEN statement.
2. If the file is explicitly opened and the name is specified as all blanks, the name is read from the command line (if available). If the command line is unavailable or contains no name, the user will usually be prompted for the name.
3. If the file is implicitly opened (with a READ or WRITE statement) the name is obtained as in method 2, described in the preceding paragraph.
4. If the file is explicitly opened and no name is specified in the OPEN statement, the file is considered a scratch or temporary file, and

an implementation-dependent name is assumed.
 (See Appendix D, "Microsoft FORTRAN Scratch Filenames," in the Microsoft FORTRAN Compiler User's Guide for the default name used by your operating system.)

The following sample program uses * files and named, external, sequential, formatted files for reading and writing. The I/O statements themselves are explained in general in Section 4.3, "I/O Statements." For details of each individual I/O statement, see the appropriate entries in Section 3.2, "Statement Directory."

```

C COPY A FILE WITH THREE COLUMNS OF INTEGERS,
C EACH 7 COLUMNS WIDE, FROM A FILE WHOSE NAME
C IS ENTERED BY THE USER TO ANOTHER FILE NAMED
C OUT.TXT, REVERSING THE POSITIONS OF THE
C FIRST AND SECOND COLUMNS.
      PROGRAM COLSWP
      CHARACTER*64 FNAME

C PROMPT TO THE SCREEN BY WRITING TO *.
      WRITE(*,900)
900      FORMAT(' INPUT FILE NAME - '\)

C READ THE FILE NAME FROM THE KEYBOARD BY
C READING FROM *.
      READ(*,910) FNAME
910      FORMAT(A)

C USE UNIT 3 FOR INPUT; ANY UNIT NUMBER EXCEPT
C 0 WILL DO.
      OPEN(3,FILE=FNAME)

C USE UNIT 4 FOR OUTPUT; ANY UNIT NUMBER EXCEPT
C 0 AND 3 WILL DO.
      OPEN(4,FILE='OUT.TXT',STATUS='NEW')

C READ AND WRITE UNTIL END OF FILE.
100      READ(3,920,END=200) I,J,K
          WRITE(4,920) J,I,K
920      FORMAT(3I7)
          GOTO 100
200      WRITE(*,910) 'Done'
          END
  
```


4.2.5 Other File Structures

The less commonly used file structures are appropriate for certain classes of applications. A very general indication of their intended uses follows:

1. If random access I/O is needed, as would probably be the case in a data base, direct access files are necessary.
2. If the data is to be both written and reread by MS-FORTRAN, unformatted files are perhaps more efficient in terms of speed, but possibly less efficient in terms of disk space. The combination of direct and unformatted files is ideal for a data base created, maintained, and accessed exclusively by MS-FORTRAN.
3. If the data must be transferred without any system interpretation, especially if all 256 possible byte values are to be transferred, unformatted I/O is necessary.

One use of unformatted I/O is in the control of a device that has a single-byte, binary interface. Formatted I/O would, in this example, interpret certain characters, such as the ASCII representation for RETURN, and fail to pass them through to the program unaltered.

The number of bytes written for an integer constant is determined by the \$STORAGE metacommand (for details, see Section 6.2.8, "The \$STORAGE Metacommand.")

4. If the data is to be transferred as in the third use described in this list, but will be read by non-FORTRAN programs, the BINARY format is recommended. Unformatted files are blocked internally, and consequently the non-FORTRAN program must be compatible with this format to interpret the data correctly. BINARY files contain only the data written to them. Backspacing over records is not possible and incomplete records cannot be read from them.

4.2.6 OLD and NEW Files

A file opened in MS-FORTRAN is either OLD or NEW, but "opened for reading" is not distinguishable from "opened for writing." Therefore, you can open OLD (existing) files and write to them, with the effect of overwriting them.

Similarly, you can alternately WRITE and READ to the same file (providing that you avoid reading beyond the end of the file, or reading unwritten records in a direct file). A WRITE to a sequential file effectively deletes any records that existed beyond the newly written record.

When a device such as the keyboard or printer is opened as a file, it normally makes no difference whether it is opened as OLD or NEW. With disk files, however, opening a file as NEW creates a new file:

1. If a previous file existed with the same name, the previous file is deleted.
2. If the new file is closed with STATUS='KEEP' or if the program terminates without doing a CLOSE operation on that file, a permanent file is created with the name given when the file was opened.

4.2.7 Limitations

Certain limitations on the use of the MS-FORTRAN I/O system are described briefly in the following list:

1. Direct files/direct device association

There are two kinds of devices: sequential and direct. The files associated with sequential devices are streams of characters; except for reading and writing, no explicit motion is allowed. The keyboard, screen, and printer are all sequential devices.

Direct devices, such as disks, have the additional task of seeking a specific location. Direct devices can be accessed either sequentially or randomly, and thus can support direct files. The MS-FORTRAN I/O system does not allow direct files on sequential devices.

2. BACKSPACE/BINARY sequential file association

There is no indication in a binary sequential file of record boundaries; therefore, a BACKSPACE operation on such files is defined as backing up by one byte. Direct files contain records of fixed, specified length, so it is possible to backspace by records on direct unformatted files. (

3. Partial READ/BINARY file

The data read from a binary file must correspond in length to the data written. Unformatted sequential files differ, in that an internal structure allows part or none of a record to be read (the unread part is skipped).

4. Side effects of functions called in I/O statements

During execution of any I/O statement, evaluation of an expression may cause a function to be called. That function call must not cause any I/O statement to be executed. (

4.3 I/O Statements

This section discusses the elements of I/O statements in general. For specific details on each of the seven I/O statements OPEN, CLOSE, READ, WRITE, BACKSPACE, ENDFILE, and REWIND, see the appropriate entries in Section 3.2, "Statement Directory," in the previous chapter.

In addition to these I/O statements, there is an I/O intrinsic function, EOF(<unit-spec>), which is described in Section 5.3.2, "Intrinsic Functions." EOF returns a logical value that indicates whether there is any data remaining in the file after the current position. (

4.3.1 Elements of I/O Statements

The various I/O statements take certain parameters and arguments that specify sources and destinations of data transfer as well as other facets of the I/O operation. The elements described in this subsection are the following:

1. unit specifier (<unit-spec>)
2. format specifier (<format-spec>)
3. input/output list (<iolist>)

4.3.1.1 The Unit Specifier

The unit specifier, <unit-spec>, can take one of the following forms in an I/O statement:

1. *
- Refers to the keyboard or screen.
2. Integer expression
- Refers to an external file with a unit number equal to the value of the expression (* is unit number zero).
3. Name of a character variable or character array element
- Refers to the internal file represented by the the variable or array element.

See Section 4.2.3, "Units," for a discussion of the difference between external and internal unit specifiers.

4.3.1.2 The Format Specifier

The format specifier, <format-spec>, can take one of the following forms in an I/O statement:

1. Statement label
- Refers to the FORMAT statement labeled by that label. For further information, see Section 3.2.18, "The FORMAT Statement."

2. Integer variable name

Refers to the FORMAT label assigned to that integer variable using the ASSIGN statement. For further information, see Section 3.2.1, "The ASSIGN Statement."

3. Character expression

The format specified is the current value of the character expression provided as the format specifier.

4. *

Indicates a list-directed I/O transfer. For further information, see Section 4.5, "List-Directed I/O."

4.3.1.3 Input/Output List

The input/output list, <iolist>, specifies the entities whose values are transferred by READ and WRITE statements. An <iolist> may be empty, but ordinarily consists of input or output entities and implied DO lists, separated by commas. An input entity can be specified in the <iolist> of a READ statement and an output entity in the <iolist> of a WRITE statement.

1. Input entities

An input entity is either a variable name, an array element name, or an array name. An array name specifies all of the elements of the array in memory sequence order.

2. Output entities

In addition to being any of the items listed as input entities, an output entity can be any other expression not beginning with the left parenthesis character "(". (The left parenthesis distinguishes implied DO lists from expressions.)

To distinguish it from an implied DO list, the following expression

$$(A+B) * (C+D)$$

can be written as:

$$+ (A+B) * (C+D)$$

3. Implied DO lists

Implied DO lists can be specified as items in the I/O list of READ and WRITE statements and have the following format:

$$(\langle \text{iolist} \rangle, \langle \text{variable} \rangle = \langle \text{expr1} \rangle, \langle \text{expr2} \rangle, [, \langle \text{expr3} \rangle])$$

$\langle \text{iolist} \rangle$ is defined the same as for elements of I/O statements (including nested implied DO lists).

$\langle \text{variable} \rangle$, $\langle \text{expr1} \rangle$, $\langle \text{expr2} \rangle$, and $\langle \text{expr3} \rangle$ are the same as defined for the DO statement. That is, $\langle \text{variable} \rangle$ is an integer variable, while $\langle \text{expr1} \rangle$, $\langle \text{expr2} \rangle$, and $\langle \text{expr3} \rangle$ are integer expressions.

In a READ statement, the DO variable (or an associated entity) must not appear as an input list item in the embedded $\langle \text{iolist} \rangle$, but may have been read in the same READ statement before the implied DO list. The embedded $\langle \text{iolist} \rangle$ is effectively repeated for each iteration of $\langle \text{variable} \rangle$ with appropriate substitution of values for the DO variable.

In the case of nested implied DO loops, the innermost (most deeply nested) loop is always executed first.

4.3.2 Carriage Control

The first character of every record transferred to a printer or other terminal device, including the console, is not printed. Instead, it is interpreted as a carriage control character. The MS-FORTRAN I/O system recognizes certain characters as carriage control characters. These characters and their effects when printed are shown in Table 4.1.

Table 4.1. Carriage Control Characters

Character	Effect
space	Advances one line.
0	Advances two lines.
1	Advances to top of next page (ignored by the console).
+ (plus)	Does not advance (allows overprinting).

Any character other than those listed in the preceding table is treated as a space and deleted from the print line. If you accidentally omit the carriage control character, the first character of the record is not printed.

4.4 Formatted I/O

If a READ or WRITE statement specifies a format, the I/O statement is considered a formatted, rather than an unformatted, I/O statement. Such a format can be specified in one of four ways, as explained previously in Section 4.3.1.2, "The Format Specifier."

Two of the four methods that refer to FORMAT statements are described in Section 3.2.18, "The FORMAT Statement"; the third is a character expression containing the format itself (see Section 4.4.2, "Edit Descriptors"); the fourth denotes that the operation is to be list-directed (see Section 4.5, "List-Directed I/O").

The following five examples are all valid and equivalent means of specifying a format:

```

      WRITE (*,990) I,J,K
990   FORMAT(1X,2I5,I3)

      ASSIGN 990 TO IFMT
990   FORMAT(1X,2I5,I3)
      WRITE (*,IFMT) I,J,K

      WRITE (*,'(1X,2I5,I3)')I,J,K

      CHARACTER*11 FMTCH
      FMTCH = '(1X,2I5,I3)'
      WRITE (*,FMTCH)I,J,K

      WRITE (*,*) I,J,K

```

The format specification must begin with a left parenthesis character and end with a matching right parenthesis character. The leading left parenthesis can be preceded by initial blank characters. Characters beyond the matching right parenthesis are ignored.

4.4.1 Interaction Between Format and I/O List

If an <iolist> contains at least one item, at least one repeatable edit descriptor must exist in the format specification. In particular, the empty edit specification, (), can be used only if no items are specified in the <iolist> (in which case a WRITE writes a zero length record and a READ skips to the next record).

Each item in the <iolist> is associated with a repeatable edit descriptor during the I/O statement execution. In contrast, the remaining format control items interact directly with the record and do not become associated with items in the <iolist>.

The items in a format specification are interpreted from left to right. Repeatable edit descriptors act as if they were present <r> times (if omitted, <r> is treated as a repeat factor of one). A format specification itself can have a repeat factor, as in:

```
10(5F1.04, 2(3x,5I3))
```

During the formatted I/O process, the "format controller" scans and processes the format items as

described in the previous paragraph. When a repeatable edit descriptor is encountered, one of the following occurs:

1. A corresponding item appears in the <iolist>, in which case the item and the edit descriptor are associated and I/O of that item proceeds under format control of the edit descriptor.
2. No corresponding item appears in the <iolist>, in which case the format controller terminates I/O. Thus, for the following statements:

```

                I=5
                WRITE (*,10) I
10              FORMAT (1X,'I= ',I5,'J= ',I5)
    
```

the output would look like this:

```

I=      5,J=
    
```

If the format controller encounters the matching final right parenthesis of the format specification and if there are no further items in the <iolist>, the format controller terminates I/O.

If, however, there are further items in the <iolist>, the file is positioned at the beginning of the next record and the format controller continues by rescanning the format, starting at the beginning of the format specification terminated by the last preceding right parenthesis.

If there is no such preceding right parenthesis, the format controller rescans the format from the beginning. Within the portion of the format rescanned, there must be at least one repeatable edit descriptor.

If the rescan of the format specification begins with a repeated nested format specification, the repeat factor indicates the number of times to repeat that nested format specification. The rescan does not change the previously set scale factor or the BN or BZ blank control in effect.

When the format controller terminates, the remaining characters of an input record are skipped or an end-of-record is written on output. An exception to this occurs when the \ edit descriptor is used. (See Section 4.4.3, "Nonrepeatable Edit Descriptors," for information on backslash editing.)

4.4.2 Edit Descriptors

Edit descriptors in FORTRAN specify the form of a record and control the editing between the characters in a record and the internal format of data. There are two types of edit descriptors: repeatable and nonrepeatable. Both are described in the following sections of this chapter.

4.4.2.1 Nonrepeatable Edit Descriptors

1. Apostrophe editing ('xxxx')

The apostrophe edit descriptor has the form of a character constant and causes the character constant to be transmitted to the output unit. Embedded blanks are significant; two adjacent apostrophes, i.e., single quotation marks, must be used to represent a single apostrophe within a character constant. Apostrophe editing cannot be used for input (READ). For an example, see "Hollerith editing (H)."

2. Hollerith editing (H)

The <n>H edit descriptor transmits the next <n> characters, with blanks counted as significant, to the output unit. Hollerith editing cannot be used for input (READ).

Examples of apostrophe and Hollerith editing:

```

C EACH WRITE OUTPUTS CHARACTERS
C BETWEEN THE SLASHES: /ABC'DEF/

C APOSTROPHE EDITING
      WRITE (*,970)
970   FORMAT (' ABC'DEF')
      WRITE (*, '(' ABC''DEF''))

C SAME OUTPUT USING HOLLERITH EDITING
      WRITE (*, '(8H ABC'DEF)')
      WRITE (*,960)
960   FORMAT (8H ABC'DEF)

```

The leading blank in each case in the preceding examples is a carriage control character to cause a line feed (carriage return) on output.

3. Positional editing (X)

On input (READ), the <n>X edit descriptor advances the file position <n> characters, skipping <n> characters. On output (WRITE), the <n>X edit descriptor writes <n> blanks, providing that further writing to the record occurs; otherwise, the <n>X descriptor results in no operation.

4. Slash editing (/)

The slash indicates the end of data transfer on the current record. On input, the file is positioned to the beginning of the next record. On output, an end-of-record is written, and the file is positioned to write on the beginning of the next record.

5. Backslash editing (\)

Normally when the format controller terminates, the end of data transmission on the current record occurs. If the last edit descriptor encountered by the format controller is a backslash (\), this automatic end-of-record is inhibited, allowing subsequent I/O statements to continue reading (or writing) from (or to) the same record. This mechanism is most commonly used to prompt to the screen and read a response from the same line, as in the following example:

```
WRITE (*,'(A\)' ) 'Input an integer --> '
READ (*,'(BN,I6)' ) I
```

The backslash edit descriptor does not inhibit the automatic end-of-record generated when reading from the * unit; input from the keyboard must always be terminated by the RETURN key. The backslash edit descriptor may not be used with internal files.

6. Scale factor editing (P)

The <k>P edit descriptor sets the scale factor for subsequent F and E edit descriptors until the next <k>P edit descriptor. At the start of each I/O statement, the scale factor is initialized to zero. The scale factor affects format editing in the following ways:

- a. On input, with F and E editing (providing that no explicit exponent exists in the field) and F output editing, the externally represented number equals the internally represented number multiplied by $10^{**<k>}$.
- b. On input, with F and E editing, the scale factor has no effect if there is an explicit exponent in the input field.
- c. On output, with E editing, the real part of the quantity is output multiplied by $10^{**<k>}$ and the exponent is reduced by $<k>$ (effectively altering the column position of the decimal point but not the value output).

7. Blank interpretation (BN and BZ)

These edit descriptors specify the interpretation of blanks in numeric input fields. The default, BZ, is set at the start of each I/O statement. This makes blanks, other than leading blanks, identical to zeros. If a BN edit descriptor is processed by the format controller, blanks in subsequent input fields are ignored unless, and until, a BZ edit descriptor is processed.

The effect of ignoring blanks is to take all the nonblank characters in the input field and treat them as if they were right-justified in the field with the number of leading blanks equal to the number of ignored blanks. For instance, the following READ statement accepts the characters shown between the slashes as the value 123 (where $<RETURN>$ indicates pressing the RETURN key):

```

      READ(*,100) I
100   FORMAT (BN,I6)

      /123      <RETURN>/
      /123      456<RETURN>/
      /  123<RETURN>/
```

Reading "short" records can temporarily invoke BN status automatically. If the total number of characters in the input record is fewer than that specified by the combination of format descriptors and $<iolist>$ elements, the

record is padded on the right with blanks to the required length, and BN editing goes into effect temporarily. Thus, the following example results in the value 123, rather than 12300:

```
      READ (*,'(I5)') I  
      /123<RETURN>/
```

The BN edit descriptor, in conjunction with the infinite blank padding at the end of formatted records, makes interactive input very convenient.

4.4.2.2 Repeatable Edit Descriptors

The I, F, E, D, and G edit descriptors are used for I/O of numeric data. The following general rules apply to all numeric edit descriptors:

1. On input, leading blanks are not significant. Other blanks are interpreted differently depending on the BN or BZ flag in effect, but all blank fields always become the value zero. Plus signs are optional. The blanks supplied by the file system to pad a record to the required size are also not significant.
2. On input with F and E editing, an explicit decimal point appearing in the input field overrides the edit descriptor specification of the decimal point position.
3. On output, the characters generated are right-justified in the field and padded by leading blanks, if necessary.
4. On output, if the number of characters produced exceeds the field width or the exponent exceeds its specified width, the entire field is filled with asterisks.

Individual descriptions of the repeatable edit descriptors follow.

1. Integer editing (I)

The edit descriptor I<w> must be associated with an <iolist> item of type INTEGER. The field is <w> characters wide. On input, an optional sign may appear in the field.

2. F real editing

The edit descriptor `F<w>.<d>` must be associated with an `<iolist>` item of type `REAL` or `REAL*8`. The field is `<w>` characters wide, with a fractional part `<d>` digits wide. The input field begins with an optional sign followed by a string of digits which may contain an optional decimal point. If the decimal point is present, it overrides the `<d>` specified in the edit descriptor; otherwise, the rightmost `<d>` digits of the string are interpreted as following the decimal point (with leading blanks converted to zeros, if necessary). Following this is an optional exponent which is either:

- a. `+` (plus) or `-` (minus) followed by an integer, or
- b. `E` followed by zero or more blanks followed by an optional sign followed by an integer.

The output field occupies `<w>` digits, `<d>` of which fall beyond the decimal point. The value output is controlled both by the `<iolist>` item and the current scale factor. The output value is rounded rather than truncated.

3. E and D real editing

The `E` edit descriptor takes one of the forms `E<w>.<d>` or `E<w>.<d>E<e>`. The `D` edit descriptor takes the form `D<w>.<d>`. All parameters and rules for the `E` edit descriptor apply to the `D` edit descriptor.

For each form, the field is `<w>` characters wide. The `<e>` has no effect on input. The input field for the `E` and `D` edit descriptors is identical to that described by an `F` edit descriptor with the same `<w>` and `<d>`.

The form of the output field depends on the scale factor (set by the `P` edit descriptor) in effect. For a scale factor of zero, the output field is a minus sign (if necessary), followed by a decimal point, followed by a string of digits, followed by an exponent field for exponent `<exp>`, of one of the forms shown in Table 4.2.

Table 4.2. Scale Factors for E and D Editing

Edit Descriptor	Absolute Value of Exponent	Form of Exponent
E<w>.<d>	exp ≤ 99	E followed by plus or minus, followed by the two-digit exponent
E<w>.<d>	99 < exp ≤ 999	Plus or minus, followed by the three-digit exponent
E<w>.<d>E<e>	exp ≤ (10**<e>)-1	E followed by plus or minus, followed by <e> digits which are the exponent with possible leading zeros
D<w>.<d>	exp ≤ 99	D followed by plus or minus, followed by the two-digit exponent
D<w>.<d>	99 < exp ≤ 999	Plus or minus, followed by the three-digit exponent

The forms E<w>.<d> and D<w>.<d> must not be used if the absolute value of the exponent to be printed exceeds 999.

The scale factor controls the decimal normalization of the printed E or D field. If the scale factor, <k>, is in the range (-d<k≤0), then the output field contains exactly <k> leading zeros after the decimal point and <d> + <k> significant digits after this. If (0<k<d+2), then the output field contains exactly <k> significant digits to the left of the decimal point and (d-k-1) places after the decimal point. Other values of <k> are errors.

4. G real editing

The G edit descriptor takes the forms $G\langle w \rangle.\langle d \rangle$ and $G\langle w \rangle.\langle d \rangle E\langle e \rangle$. For either form, the input field is $\langle w \rangle$ characters wide, with a fractional part consisting of $\langle d \rangle$ digits. If the scale factor is greater than one, the exponent part consists of $\langle e \rangle$ digits.

G input editing is the same as F input editing.

G output editing is dependent on the magnitude of the data being edited. Table 4.3 illustrates the output equivalent for the magnitude of data.

Table 4.3. Data Conversion Equivalents

Data Magnitude	Conversion Equivalent
$M < 0.1$	$E\langle w \rangle.\langle d \rangle$
$0.1 \leq M < 1$	$F(\langle w \rangle - n).\langle d \rangle, n('b')$ [1, 2]
$1 \leq M < 10$	$F(\langle w \rangle - n).(\langle d \rangle - 1), n('b')$
\cdot	\cdot
\cdot	\cdot
$10^{**}(\langle d \rangle - 2) \leq M$ $< 10^{**}(\langle d \rangle - 1)$	$F(\langle w \rangle - n).1, n('b')$
$10^{**}(\langle d \rangle - 1) \leq M$ $< 10^{**}\langle d \rangle$	$F(\langle w \rangle - n).0, n('b')$
$M \geq 10^{**}\langle d \rangle$	$E\langle w \rangle.\langle d \rangle$

Notes for Table 4.3:

1. 'b' represents a blank character.
2. n is 4 for $G\langle w \rangle.\langle d \rangle$;
n is $\langle e \rangle + 2$ for $G\langle w \rangle.\langle d \rangle E\langle e \rangle$.

5. Logical editing (L)

The edit descriptor takes the form L<w>, indicating that the field is <w> characters wide. The <iolist> element associated with an L edit descriptor must be of type LOGICAL. On input, the field consists of optional blanks, followed by an optional decimal point, followed by T (for .TRUE.) or F (for .FALSE.). Any further characters in the field are ignored, but accepted on input, so that .TRUE. and .FALSE. are valid inputs. On output, w-1 blanks are followed by either T or F, as appropriate.

6. Character editing (A)

The forms of the edit descriptor are A or A<w>. In the first form, A acquires an implied field width, <w>, from the number of characters in the <iolist> associated item. The <iolist> item must be of type CHARACTER if it is to be associated with an A or A<w> edit descriptor.

On input, if <w> exceeds or equals the number of characters in the <iolist> element, the rightmost characters of the input field are used as the input characters; otherwise, the input characters are left-justified in the input <iolist> item and trailing blanks are provided.

If the number of characters input is not equal to <w>, then the input field will be blankfilled or truncated on the right to the length of <w> before being transmitted to the <iolist> item. For example, if the following program fragment is executed,

```
CHARACTER*10 C
READ(*,'(A15)') C
```

and the following thirteen characters are typed in at the keyboard,

```
'ABCDEFGHIJKLM'
```

the input field will be filled to fifteen characters:

```
'ABCDEFGHIJKLM '
```


Then the rightmost ten characters will be transmitted to the <iolist> element C:

'FGHIJKLM '

On output, if <w> exceeds the characters produced by the <iolist> item, leading blanks are provided; otherwise, the leftmost <w> characters of the <iolist> item are output.

4.5 List-Directed I/O

A list-directed record is a sequence of values and value separators.

Each value in a list-directed record is one of the following:

1. a constant
2. a null value
3. either a constant or a null value multiplied by an unsigned, nonzero, integer constant; that is, $r*c$ (r successive appearances of the constant c) or $r*$ (r successive null values). Except in string constants, none of these may have embedded blanks.

Each value separator in a list-directed record is one of the following:

1. a comma, optionally preceded or followed by one or more contiguous blanks
2. a slash, optionally preceded or followed by one or more contiguous blanks
3. one or more contiguous blanks between two constants, or after the last constant

4.5.1 List-Directed Input

Except as noted in the following list, input forms acceptable to format specifications for a given type are also acceptable for list-directed formatting.

The form of the input value must be acceptable for the type of the input list item. Never use blanks as zeros. Only use embedded blanks within character constants, as specified in the following list. Note that the end-of-record has the effect of a blank, except when it appears within a character constant.

1. Real or double precision constants

A real or double precision constant must be a numeric input field; that is, a field suitable for F editing. It is assumed to have no fractional digits unless there is a decimal point within the field.

2. Logical constants

A logical constant must not include either slashes or commas among the optional characters permitted for L editing.

3. Character constants

A character constant is a nonempty string of characters, enclosed in single quotation marks. Each single quotation mark within a character constant must be represented by two single quotation marks, with no intervening blank or end-of-record.

Character constants may be continued from the end of one record to the beginning of the next; the end of the record doesn't cause a blank or other character to become part of the constant. The constant may be continued on as many records as needed and may include the characters blank, comma, and slash.

If the length $\langle n \rangle$ of the list item is less than or equal to the length $\langle m \rangle$ of the character constant, the leftmost $\langle n \rangle$ characters of the latter are transmitted to the list item. If $\langle n \rangle$ is greater than $\langle m \rangle$, the constant is transmitted to the leftmost $\langle m \rangle$ characters of the list item.

The remaining $\langle n \rangle$ minus $\langle m \rangle$ characters of the list item are filled with blanks. The effect is the same as if the constant were assigned to the list item in a character assignment statement.

4. Null values

You can specify a null value in one of three ways:

- a. no characters between successive value separators
- b. no characters preceding the first value separator in the first record read by each execution of a list-directed input statement
- c. the r* form (described at the beginning of Section 4.5, "List-Directed I/O")

A null value has no effect on the definition status of the corresponding input list item. If the input list item is defined, it retains its previous value; if it is undefined, it remains so.

A slash encountered as a value separator during execution of a list-directed input statement stops execution of that statement after the assignment of the previous value. Any further items in the input list are treated as if they were null values.

5. Blanks

All blanks in a list-directed input record are considered to be part of some value separator, except for the following:

- a. blanks embedded in a character constant
- b. leading blanks in the first record read by each execution of a list-directed input statement (unless immediately followed by a slash or comma)

4.5.2 List-Directed Output

The form of the values produced is the same as required for input, except as noted in the following list.

1. New records are created as necessary, but except for character constants, neither the end of a record nor blanks will occur within a constant.
2. Logical output constants are T for the value true and F for the value false.
3. Integer output constants are produced with the effect of an I12 edit descriptor.
4. Real and double precision constants are produced with the effect of either an F or an E edit descriptor, depending on the value of x in the following range:

$$10^{**0} \leq x \leq 10^{**7}$$

- a. If x is within the range, the constant is produced using 0PF16.7 for single precision and 0PF23.14 for double precision.
 - b. If x is outside the range, the constant is produced using 1PE14.6 for single precision and 1PE21.13 for double precision.
5. Character constants produced have the following characteristics:
 - a. They are not delimited by apostrophes (single quotation marks).
 - b. They are neither preceded nor followed by a value separator.
 - c. Each internal apostrophe (single quotation mark) is represented by one externally.
 - d. A blank character is inserted at the start of any record that begins with the continuation of a character constant from the preceding record.

6. Slashes, as value separators, and null values are not produced by list-directed formatting.
7. In order to provide carriage control when the record is printed, each output record begins with a blank character.

(

(

(

CHAPTER 5
PROGRAMS, SUBROUTINES, AND FUNCTIONS

5.1	Main Program	147
5.2	Subroutines	147
5.3	Functions	148
5.3.1	External Functions	148
5.3.2	Intrinsic Functions	149
5.3.3	Statement Functions	155
5.4	Arguments	155

(

(

(

As described in Section 1.2, "Programs and Compilable Parts of Programs," a program unit is either a main program, a subroutine, or a function. Functions and subroutines are collectively called subprograms, or procedures. The PROGRAM, SUBROUTINE, and FUNCTION statements, as well as the statement function statement, are described in detail in Section 3.2, "Statement Directory." Related information is provided in the entries for the CALL and RETURN statements.

This chapter supplements the discussion of these individual statements with information on types of functions and a description of the relationship between formal and actual arguments in a function or subroutine call.

5.1 Main Program

A main program is any program unit that does not have a FUNCTION or SUBROUTINE statement as its first statement. The first statement of a main program may be a PROGRAM statement. If the main program does not have a program statement, it will be assigned the name MAIN. The name MAIN then cannot be used to name any other global entity.

The execution of a program always begins with the first executable statement in the main program. Consequently, there must be precisely one main program in every executable program.

For further information about programs, see Section 3.2.30, "The PROGRAM Statement."

5.2 Subroutines

A subroutine is a program unit that can be called from other program units with a CALL statement. When invoked, a subroutine performs the set of actions defined by its executable statements and then returns control to the statement immediately following the one that called it.

A subroutine does not directly return a value, although values can be passed back to the calling program unit via arguments or common variables.

For further information about subroutines, see Section 3.2.37, "The SUBROUTINE Statement," and Section 3.2.4, "The CALL Statement."

5.3 Functions

A function is referred to in an expression and returns a value that is used in the computation of that expression. There are three kinds of functions:

1. external functions
2. intrinsic functions
3. statement functions

Each of these is described in more detail in the following sections.

Reference to a function may appear in an arithmetic or logical expression. When the function reference is executed, the function is evaluated and the resulting value used as an operand in the expression that contains the function reference. The format of a function reference is as follows:

<fname> ([<arg> [, <arg>] ...])

<fname> is the user-defined name of an external, intrinsic, or statement function.

<arg> is an actual argument.

The rules for arguments for functions are identical to those for subroutines and are described in Section 3.2.4, "The CALL Statement." Some additional restrictions that apply for intrinsic functions and for statement functions are described in Section 5.3.2, "Intrinsic Functions," and Section 5.3.3, "Statement Functions," respectively.

5.3.1 External Functions

An external function is specified by a function program unit. It begins with a FUNCTION statement and ends with an END statement. It may contain any kind of statement other than a PROGRAM statement, FUNCTION statement, or a SUBROUTINE statement.

5.3.2 Intrinsic Functions

Intrinsic functions are predefined by the MS-FORTRAN language and available for use in an MS-FORTRAN program. Table 5.1 gives the name, definition, argument type, and function type for all of the intrinsic functions available in MS-FORTRAN, with additional notes following the table.

An IMPLICIT statement cannot alter the type of an intrinsic function. For those intrinsic functions that allow several types of arguments, all arguments in a single reference must be of the same type.

An intrinsic function name can appear in an INTRINSIC statement. An intrinsic function name also can appear in a type statement, but only if the type is the same as the standard type of that intrinsic function.

Arguments to certain intrinsic functions are limited by the definition of the function being computed. For example, the logarithm of a negative number is mathematically undefined, and therefore not permitted.

All angles in Table 5.1 are expressed in radians. All arguments in an intrinsic function reference must be of the same type. X and Y are real, I and J are integer, and C, C1, and C2 are character values. Numbers in square brackets in column 1 refer to the notes following the table.

Furthermore, REAL is equivalent to REAL*4, DOUBLE PRECISION is equivalent to REAL*8. If the specified type of the argument is INTEGER, the type may be INTEGER*2 or INTEGER*4. If the specified type of the function is INTEGER, the type will be the default integer determined by the \$STORAGE metacommand. (For further information, see Section 6.2.8, "The \$STORAGE Metacommand.")

Table 5.1. Intrinsic Functions

Name	Definition	Type of Argument	Type of Function
Type Conversion			
INT(X) [1]	Convert to integer	REAL*4	Int
		Int	Int
IFIX(X)	Convert to integer	REAL*4	Int
IDINT(2)	Convert to integer	REAL*8	Int
REAL(X) [2]	Convert to REAL*4	Int	REAL*4
		REAL*4	REAL*4
FLOAT(I)	Convert to REAL*4	Int	REAL*4
ICHAR(C) [3]	Convert to integer	Char	Int
CHAR(X) [3]	Convert to character	Int	Char
SNGL(X)	Convert to REAL*4	REAL*8	REAL*4
DBLE(X) [4]	Convert to REAL*8	Int	REAL*8
		REAL*4	REAL*8
		REAL*8	REAL*8
Truncation			
AINT(X)	Truncate to REAL*4	REAL*4	REAL*4
DINT(X)	Truncate to REAL*8	REAL*8	REAL*8
Nearest Whole Number			
ANINT(X)	Round to REAL*4	REAL*4	REAL*4
DNINT(X)	Round to REAL*8	REAL*8	REAL*8
Nearest Integer			
NINT(X)	Round to integer	REAL*4	Int
IDNINT(X)	Round to integer	REAL*8	Int
Absolute Value			
IABS(I)	Int absolute	Int	Int
ABS(X)	REAL*4 absolute	REAL*4	REAL*4
DABS(X)	REAL*8 absolute	REAL*8	REAL*8
Remaindering			
MOD(I,J)	Int remainder	Int	Int
AMOD(X,Y)	REAL*4 remainder	REAL*4	REAL*4
DMOD(X,Y)	REAL*8 remainder	REAL*8	REAL*8

Table 5.1. Intrinsic Functions (continued)

Name	Definition	Type of Argument	Type of Function
Transfer of Sign			
ISIGN(I,J)	Int transfer	Int	Int
SIGN(X,Y)	REAL*4 transfer	REAL*4	REAL*4
DSIGN(X,Y)	REAL*8 transfer	REAL*8	REAL*8
Positive Difference [5]			
IDIM(I,J)	Int difference	Int	Int
DIM(X,Y)	REAL*4 difference	REAL*4	REAL*4
DDIM(X,Y)	REAL*8 difference	REAL*8	REAL*8
Choosing Largest Value			
MAX0(I,J,...)	Int maximum	Int	Int
AMAX1(X,Y,...)	REAL*4 maximum	REAL*4	REAL*4
AMAX0(I,J,...)	REAL*4 maximum	Int	REAL*4
MAX1(X,Y,...)	Int maximum	REAL*4	Int
DMAX1(X,Y,...)	REAL*8 maximum	REAL*8	REAL*8
Choosing Smallest Value			
MIN0(I,J,...)	Int minimum	Int	Int
AMIN1(X,Y,...)	REAL*4 minimum	REAL*4	REAL*4
AMIN0(I,J,...)	REAL*4 minimum	Int	REAL*4
MIN1(X,Y,...)	Int minimum	REAL*4	Int
DMIN1(X,Y,...)	REAL*8 minimum	REAL*8	REAL*8
REAL*8 Product			
DPROD	REAL*8 product	REAL*4	REAL*8
Square Root			
SQRT	Square root	REAL*4	REAL*4
DSQRT	REAL*8 square root	REAL*8	REAL*8
Exponential			
EXP(X)	REAL*4 e to power	REAL*4	REAL*4
DEXP(X)	REAL*8 e to power	REAL*8	REAL*8

Table 5.1. Intrinsic Functions (continued)

Name	Definition	Type of Argument	Type of Function
Natural Logarithm			
ALOG (X)	Nat'l log of REAL*4	REAL*4	REAL*4
DLOG (X)	Nat'l log of REAL*8	REAL*8	REAL*8
Common Logarithm			
ALOG10 (X)	Common log of REAL*4	REAL*4	REAL*4
DLOG10 (X)	Common log of REAL*8	REAL*8	REAL*8
Sine			
SIN (X)	REAL*4 sine	REAL*4	REAL*4
DSIN (X)	REAL*8 sine	REAL*8	REAL*8
Cosine			
COS (X)	REAL*4 cosine	REAL*4	REAL*4
DCOS (X)	REAL*8 cosine	REAL*8	REAL*8
Tangent			
TAN (X)	REAL*4 tangent	REAL*4	REAL*4
DTAN (X)	REAL*8 tangent	REAL*8	REAL*8
Arc Sine			
ASIN (X)	REAL*4 arc sine	REAL*4	REAL*4
DASIN (X)	REAL*8 arc sine	REAL*8	REAL*8
Arc Cosine			
ACOS (X)	REAL*4 arc cosine	REAL*4	REAL*4
DACOS (X)	REAL*8 arc cosine	REAL*8	REAL*8
Arc Tangent			
ATAN (X)	REAL*4 arc tangent	REAL*4	REAL*4
DATAN (X)	REAL*8 arc tangent	REAL*8	REAL*8
ATAN2 (X,Y)	REAL*4 arctan of X/Y	REAL*4	REAL*4
DATAN2 (X,Y)	REAL*8 arctan of X/Y	REAL*8	REAL*8

Table 5.1. Intrinsic Functions (continued)

Name	Definition	Type of Argument	Type of Function
Hyperbolic Sine			
SINH (X)	REAL*4 hyperbolic sine	REAL*4	REAL*4
DSINH (X)	REAL*8 hyperbolic sine	REAL*8	REAL*8
Hyperbolic Cosine			
COSH (X)	REAL*4 hyperbolic cosine	REAL*4	REAL*4
DCOSH (X)	REAL*8 hyperbolic cosine	REAL*8	REAL*8
Hyperbolic Tangent			
TANH (X)	REAL*4 hyperbolic tangent	REAL*4	REAL*4
DTANH (X)	REAL*8 hyperbolic tangent	REAL*8	REAL*8
Lexically Greater Than or Equal			
LGE (C1,C2)	1st argument greater than or equal to 2nd	Char	Logical
Lexically Greater Than			
LGT (C1,C2)	1st argument greater than 2nd	Char	Logical
Lexically Less Than or Equal [6]			
LLE (C1,C2)	1st argument less than or equal to 2nd	Char	Logical
Lexically Less Than			
LLT (C1,C2)	1st argument less than 2nd	Char	Logical
End of File [7]			
EOF (X)	Int end of file	Int	Logical

Notes for Table 5.1:

1. For X of type INTEGER, INT(X)=X. For X of type REAL or REAL*8, if X is greater than or equal to zero, then INT(X) is the largest integer not greater than X, and if X is less than zero, then INT(X) is the most negative integer not less than X. For X of type REAL, IFIX(X) is the same as INT(X).
2. For X of type REAL, REAL(X)=X. For X of type INTEGER or REAL*8, REAL(X) is as much precision of the significant part of X as a real datum can contain. For X of type INTEGER, FLOAT(X) is the same as REAL(X).
3. For X of type REAL*8, DBLE(X)=X. For X of type INTEGER or REAL, DBLE(X) is as much precision of the significant part of X as a double precision datum can contain.
4. ICHAR converts a character value into an integer value. The integer value of a character is the ASCII internal representation of that character, and is in the range 0 to 255. For any two characters, c1 and c2, (c1 .LE. c2) is .TRUE. if and only if (ICHAR(c1) .LE. ICHAR(c2)) is .TRUE.

CHAR<i> returns the <i>th character in the collating sequence. The value is of type CHARACTER, length one, while <i> must be an integer expression whose value is in the range 0 <= <i> <= 255.

ICHAR(CHAR<i>) = <i> for 0 <= <i> <= 255.

CHAR(ICHAR(c)) = c for any character c in the character set.

5. DIM(X,Y) is X-Y if X>Y, zero otherwise.
6. LGE(X,Y) returns the value .TRUE. if X = Y or if X follows Y in the ASCII collating sequence; otherwise it returns .FALSE.

LGT(X,Y) returns .TRUE. if X follows Y in the ASCII collating sequence; otherwise it returns .FALSE.

LLE(X,Y) returns .TRUE. if X = Y or if X precedes Y in the ASCII collating sequence; otherwise it returns .FALSE.

LLT(X,Y) returns .TRUE. if X precedes Y in the ASCII collating sequence; otherwise it returns .FALSE.

If the operands are of unequal length, the shorter operand is considered to be blankfilled on the right to the length of the longer.

7. EOF(X) returns the value .TRUE. if the unit specified by its argument is at or past the end-of-file record; otherwise it returns .FALSE. The value of X must correspond to an open file, or to zero which indicates the screen or keyboard device.

5.3.3 Statement Functions

A statement function is defined by a single statement and is similar in form to an assignment statement. A statement function statement can only appear after the specification statements and before any executable statements in the program unit in which it appears.

A statement function is not an executable statement, since it is not executed in order as the first statement in its particular program unit. Rather, the body of a statement function serves to define the meaning of the statement function. It is executed, as any other function, by the execution of a function reference in an expression.

For information on the syntax and use of a statement function statement, see Section 3.2.35, "The Statement Function Statement."

5.4 Arguments

A formal argument is the name by which the argument is known within a function or subroutine; an actual argument is the specific variable, expression, array, etc., passed to the procedure in question at any specific calling location. The relationship between formal and actual arguments in a function or subroutine call is discussed in detail in the following paragraphs.

Arguments pass values into and out of procedures by reference. The number of actual arguments must be the same as the number of formal arguments, and the corresponding types must agree.

Upon entry to a subroutine or function, the actual arguments are associated with the formal arguments, much as an EQUIVALENCE statement associates two or more arrays or variables, and COMMON statements in two or more program units associate lists of variables. This association remains in effect until execution of the subroutine or function is terminated. Thus, assigning a value to a formal argument during execution of a subroutine or function may alter the value of the corresponding actual argument.

If an actual argument is a constant, function reference, or an expression other than a simple variable, assigning a value to the corresponding formal argument is not permitted, and can have some strange side effects. In particular, assigning a value to a formal argument of type CHARACTER, when the actual argument is a literal, can produce anomalous behavior.

If an actual argument is an expression, it is evaluated immediately prior to the association of formal and actual arguments. If an actual argument is an array element, its subscript expressions are evaluated just prior to the association, and remain constant throughout the execution of the procedure, even if they contain variables that are redefined during the execution of the procedure.

A formal argument that is a variable can be associated with an actual argument that is a variable, an array element, or an expression.

A formal argument that is an array can be associated with an actual argument that is an array or an array element. The number and size of dimensions in a formal argument may be different from those of the actual argument, but any reference to the formal array must be within the limits of the memory sequence in the actual array. While a reference to an element outside these bounds is not detected as an error in a running MS-FORTRAN program, the results are unpredictable.

A formal argument may also be associated with an external subroutine, function, or intrinsic function if it is used in the body of the procedure as a subroutine or function reference, or if it appears in an EXTERNAL statement.

A corresponding actual argument must be an external subroutine or function, declared with the EXTERNAL statement, or an intrinsic function permitted to be associated with a formal procedure argument. The intrinsic function must have been declared with an INTRINSIC statement in the program unit where it is used as an actual argument.

All intrinsic functions, except the following, may be associated with formal procedure arguments:

INT	CHAR	AMAX0
IFIX	LGE	MAX1
IDINT	LGT	MIN0
FLOAT	LLE	AMIN1
SNGL	LLT	DMIN1
REAL	MAX0	AMIN0
DBLE	AMAX1	MIN1
ICHAR	DMAX1	

(

(

(

CHAPTER 6
THE MICROSOFT FORTRAN METACOMMANDS

6.1	Overview	161
6.2	Metacommand Directory	163
6.2.1	The \$DEBUG and \$NODEBUG Metacommands	163
6.2.2	The \$DO66 Metacommand	164
6.2.3	The \$INCLUDE Metacommand	165
6.2.4	The \$LINESIZE Metacommand	166
6.2.5	The \$LIST and \$NOLIST Metacommands	167
6.2.6	The \$PAGE Metacommand	168
6.2.7	The \$PAGESIZE Metacommand	169
6.2.8	The \$STORAGE Metacommand	170
6.2.9	The \$STRICT and \$NOTSTRICT Metacommands	171
6.2.10	The \$SUBTITLE Metacommand	172
6.2.11	The \$TITLE Metacommand	173

(

(

(

6.1 Overview

Metacommands are directives that order the MS-FORTRAN Compiler to process MS-FORTRAN source text in a specific way. MS-FORTRAN metacommands are described briefly in Table 6.1 and discussed in more detail in the remainder of the chapter.

Table 6.1. The Microsoft FORTRAN Metacommands

Metacommand	Action
\$DEBUG	Turns on runtime checking for arithmetic operations and assigned GOTO. \$NODEBUG turns checking off.
\$DO66	Causes DO statements to have FORTRAN 66 semantics.
\$INCLUDE:<file>	Directs compiler to proceed as if <file> were inserted at that point.
\$LINESIZE:<n>	Makes subsequent pages of listing <n> columns wide.
\$LIST	Sends subsequent listing information to the listing file. \$NOLIST stops generation of listing information.
\$PAGE	Starts new page of listing.
\$PAGESIZE:<n>	Makes subsequent pages of listing <n> lines long.
\$STORAGE:<n>	Allocates <n> bytes of memory to all LOGICAL or INTEGER variables in source.
\$STRICT	Disables MS-FORTRAN features not in 1977 subset or full language standard. \$NOTSTRICT enables them.
\$SUBTITLE:'<sub>'	Gives subtitle for subsequent pages of listing.
\$TITLE:'<title>'	Gives title for subsequent pages of listing.

Metacommands can be intermixed with MS-FORTRAN source text within an MS-FORTRAN source program; however, they are not part of the standard FORTRAN language. Any line of input to the MS-FORTRAN Compiler that begins with a "\$" character in column one is interpreted as a metacommand and must conform to one of the following formats. (

A metacommand and its arguments (if any) must fit on a single source line; continuation lines are not permitted. Also, blanks are significant, so that the following pair is not equivalent:

```
$S  TRICT
$STRICT
```


6.2 Metacommand Directory

The remainder of this chapter is an alphabetical directory of available MS-FORTRAN metacommands.

6.2.1 The \$DEBUG and \$NODEBUG Metacommands

Syntax \$[NO]DEBUG

Purpose Directs the compiler to test all subsequent arithmetic operations for overflow and division by zero, test assigned GOTO values against the allowable list in an assigned GOTO statement, and provide the runtime error-handling system with source filenames and line numbers. A runtime error occurs if one of these conditions is detected. If any runtime error occurs, the source line and filename are displayed on the console.

Remarks The metacommand can appear anywhere in a program.

The default value of the pair of metacommands, \$DEBUG and \$NODEBUG, is \$NODEBUG.

6.2.2 The \$DO66 Metacommand

Syntax \$DO66

Purpose Causes DO statements to have FORTRAN 66 semantics.

Remarks \$DO66 must precede the first declaration or executable statement of the source file in which it occurs.

The FORTRAN 66 semantics are as follows:

1. All DO statements are executed at least once.
2. Extended range is permitted; that is, control may transfer into the syntactic body of a DO statement. The range of the DO statement is thereby extended to logically include any statement that may be executed between a DO statement and its terminal statement. However, the transfer of control into the range of a DO statement prior to the execution of the DO statement or following the final execution of its terminal statement is invalid.

If a program contains no \$DO66 metacommand, the default is to FORTRAN 77 semantics, as follows:

1. DO statements may be executed zero times, if the initial control variable value exceeds the final control variable value (or the corresponding condition for a DO statement with negative increment).
2. Extended range is invalid; that is, control may not transfer into the syntactic body of a DO statement. (Both standards do permit transfer of control out of the body of a DO statement.)

6.2.3 The \$INCLUDE Metacommand

Syntax \$INCLUDE: '<file>'

Purpose Directs the compiler to proceed as though the specified file were inserted at the point of the \$INCLUDE.

Remarks <file> is a valid file specification as described for your operating system.

At the end of the included file, the compiler resumes processing the original source file at the line following \$INCLUDE.

The compiler imposes no limit on nesting levels for \$INCLUDE metacommands. \$INCLUDE metacommands are particularly useful for guaranteeing that several modules use the same declaration for a COMMON block.

6.2.4 The \$LINESIZE Metacommand

Syntax \$LINESIZE: <n>

Purpose Formats subsequent pages of the listing <n>
 columns wide.

Remarks <n> is any positive integer.

If a program contains no \$LINESIZE
metacommand, a default line size of 80
characters is assumed. The minimum line
size is 40 characters, the maximum is 132
characters.

6.2.5 The \$LIST and \$NOLIST Metacommands

Syntax \$[NO]LIST

Purpose Sends subsequent listing information to the listing file specified when starting the compiler. If no listing file is specified in response to the compiler prompt, the metacommand has no effect. \$NOLIST directs that subsequent listing information be discarded.

Remarks \$LIST and \$NOLIST can appear anywhere in a source file.

The default condition for the pair of metacommands, \$LIST and \$NOLIST, is \$LIST.

6.2.6 The \$PAGE Metacommand

Syntax \$PAGE

Purpose Starts a new page of the listing.

Remarks If the first character of a line of source text is the ASCII form feed character (hexadecimal code 0Ch), it is considered as equivalent to the occurrence of a \$PAGE metacommand at that point.

6.2.7 The \$PAGESIZE Metacommand

Syntax	\$PAGESIZE: <n>
Purpose	Formats subsequent pages of the listing <n> lines high.
Remarks	<p><n> is any positive integer equal to or greater than 15.</p> <p>If a program contains no \$PAGESIZE metacommand, a default page size of 66 lines is assumed.</p>

6.2.8 The \$STORAGE Metacommand

Syntax \$STORAGE: <n>

Purpose Allocates <n> bytes of memory for all variables declared in the source file as INTEGER or LOGICAL.

Remarks <n> is either 2 or 4. Use a value of 2 for code that defaults to 16-bit arithmetic. See also the important note on performance issues in Section 2.3, "Data Types."

\$STORAGE does not affect the allocation of memory for variables declared with an explicit length specification, for example, as INTEGER*2 or LOGICAL*4.

If several files of a source program are compiled and linked together, you should be particularly careful that they are consistent in their allocation of memory for variables (such as actual and formal parameters) referred to in more than one module.

The \$STORAGE metacommand must precede the first declaration statement of the source file in which it occurs.

If a program contains no \$STORAGE metacommand, a default allocation of 4 bytes is used. This default results in INTEGER, LOGICAL, and REAL variables being allocated the same amount of memory, as required by the FORTRAN 77 standard.

6.2.9 The \$STRICT and \$NOTSTRICT Metacommands

Syntax \$[NOT]STRICT

Purpose \$STRICT disables the specific MS-FORTRAN features not found in the FORTRAN 77 subset or full language standard.

Remarks The \$NOTSTRICT metacommand enables these MS-FORTRAN features, which are the following:

1. Character expressions may be assigned to noncharacter variables.
2. Character and noncharacter expressions may be compared.
3. Character and noncharacter variables are allowed in the same COMMON block.
4. Character and noncharacter variables may be equivalenced.
5. Noncharacter variables may be initialized with character data.

\$STRICT and \$NOTSTRICT can appear anywhere in a source file.

The default condition for the pair of metacommands, \$STRICT and \$NOTSTRICT, is \$NOTSTRICT.

6.2.10 The \$SUBTITLE Metacommand

Syntax \$SUBTITLE: '<subtitle>'

Purpose Assigns the specified subtitle for subsequent pages of the source listing (until overridden by another \$SUBTITLE metacommand).

Remarks <subtitle> is any valid character constant. The maximum length is 40 characters.

If a program contains no \$SUBTITLE metacommand, the subtitle is a null string.

6.2.11 The \$TITLE Metacommand

Syntax \$TITLE: '<title>'

Purpose Assigns the specified title for subsequent pages of the listing (until overridden by another \$TITLE metacommand).

Remarks <title> is any valid character constant.
The maximum length is 40 characters.

If a program contains no \$TITLE metacommand, the title is a null string.

(

(

(

APPENDICES

A	Microsoft FORTRAN and ANSI Subset FORTRAN	177
B	ASCII Character Codes	183
C	Error Messages	185

(

(

(

APPENDIX A

MICROSOFT FORTRAN AND ANSI SUBSET FORTRAN

A.1	Full Language Features	179
A.2	Extensions to the Standard	181

(

(

(

This appendix describes how MS-FORTRAN differs from the standard subset language. The ANSI standard defines two levels, full FORTRAN and subset FORTRAN. MS-FORTRAN is a superset of the latter. The differences between MS-FORTRAN and the standard subset FORTRAN fall into two general categories: full language features and extensions to the standard.

A.1 Full Language Features

Several features from the full language are included in this implementation. In all cases, a program written to comply with the subset restrictions compiles and executes properly, since the full language includes the subset constructs.

1. Subscript expressions

The subset does not allow function calls or array element references in subscript expressions; however, these are allowed in the full language and in this implementation.

2. DO variable expressions

The subset restricts expressions that define the limits of a DO statement; the full language does not. MS-FORTRAN also allows full integer expressions in DO statement limit computations. Similarly, arbitrary integer expressions are allowed in implied DO loops associated with READ and WRITE statements.

3. Unit I/O number

MS-FORTRAN allows an I/O unit to be specified by an integer expression, as does the full language.

4. Expressions in input/output list <iolist>

The subset does not allow expressions to appear in an <iolist>, whereas the full language does allow expressions in the <iolist> of WRITE statements. MS-FORTRAN allows expressions in the <iolist> of a WRITE statement providing that the expressions do not begin with an initial left parenthesis.

Note that an expression like $(A+B)*(C+D)$ can be specified in an output list as $+(A+B)*(C+D)$. Doing so does not generate any extra code to evaluate the leading plus sign.

5. Double precision

The subset does not allow double precision real numbers; MS-FORTRAN provides for them as in the full language.

6. Edit descriptors

MS-FORTRAN allows for D and G edit descriptors as in the full language.

7. Expression in computed GOTO

MS-FORTRAN allows an expression for the selector of a computed GOTO, consistent with the full, rather than the subset, language.

8. Generalized I/O

MS-FORTRAN allows both sequential and direct access files to be either formatted or unformatted. The subset language requires direct access files to be unformatted and sequential files to be formatted.

MS-FORTRAN also includes the following:

- a. an augmented OPEN statement that takes additional parameters not included in the subset (see Section 3.2.28, "The OPEN Statement")
- b. a form of the CLOSE statement, which is not included in the subset (see Section 3.2.5, "The CLOSE Statement")
- c. END=, ERR=, STATUS=, and FILE= specifiers on I/O statements

9. List-directed I/O

MS-FORTRAN provides for list-directed I/O as described in the full language standard.

A.2 Extensions to the Standard

The implemented language also has several minor extensions to the full language standard.

1. User-defined names greater than six characters are allowed, although only the first six characters are significant.
2. Tabs in source files are allowed. See Section 2.1.3, "Tabs," for details.
3. Metacommands, or compiler directives, have been added to allow the programmer to communicate certain information to the compiler. The metacommand line is characterized by a dollar sign (\$) appearing in column 1. A metacommand line may appear any place that a comment line can appear, although certain metacommands are restricted as to their location within a program (see Section 2.2.4, "Statement Definition and Order").

A metacommand line conveys certain compile time information about the nature of the current compilation to the MS-FORTRAN Compiler. Metacommands are described in Chapter 6, "The Microsoft FORTRAN Metacommands."

4. The standard is relaxed when the \$NOTSTRICT metacommand is in effect. This relaxation allows, for example, such MS-FORTRAN features as assignment of character to any variable type and initialization of any variable with character data. See Section 6.2.9, "The \$STRICT and \$NOTSTRICT Metacommands," for a complete list of these features.
5. The backslash (\) edit control character can be used in format specifications to inhibit normal advancement to the next record associated with the completion of a READ or WRITE statement. This is particularly useful

when prompting to an interactive device, such as the screen, so that a response can appear on the same line as the prompt.

6. An end-of-file intrinsic function, EOF, is provided. The function accepts a unit specifier as an argument and returns a logical value that indicates whether the specified unit is at its end-of-file.
7. Both upper and lowercase source input are allowed. In most contexts, lowercase characters are treated as indistinguishable from their uppercase counterparts. However, lowercase is significant in character constants and Hollerith fields.
8. Binary files are similar to unformatted sequential files except that they have no internal structure. This allows the program to create or read files with arbitrary contents, which is particularly useful for files created by or intended for programs written in languages other than FORTRAN.

APPENDIX B
ASCII CHARACTER CODES

Dec	Hex	CHR	Dec	Hex	CHR
000	00H	NUL	032	20H	SPACE
001	01H	SOH	033	21H	!
002	02H	STX	034	22H	"
003	03H	ETX	035	23H	#
004	04H	EOT	036	24H	\$
005	05H	ENQ	037	25H	%
006	06H	ACK	038	26H	&
007	07H	BEL	039	27H	'
008	08H	BS	040	28H	(
009	09H	HT	041	29H)
010	0AH	LF	042	2AH	*
011	0BH	VT	043	2BH	+
012	0CH	FF	044	2CH	,
013	0DH	CR	045	2DH	-
014	0EH	SO	046	2EH	.
015	0FH	SI	047	2FH	/
016	10H	DLE	048	30H	0
017	11H	DC1	049	31H	1
018	12H	DC2	050	32H	2
019	13H	DC3	051	33H	3
020	14H	DC4	052	34H	4
021	15H	NAK	053	35H	5
022	16H	SYN	054	36H	6
023	17H	ETB	055	37H	7
024	18H	CAN	056	38H	8
025	19H	EM	057	39H	9
026	1AH	SUB	058	3AH	:
027	1BH	ESCAPE	059	3BH	;
028	1CH	FS	060	3CH	<
029	1DH	GS	061	3DH	=
030	1EH	RS	062	3EH	>
031	1FH	US	063	3FH	?

Dec	Hex	CHR	Dec	Hex	CHR
064	40H	@	096	60H	'
065	41H	A	097	61H	a
066	42H	B	098	62H	b
067	43H	C	099	63H	c
068	44H	D	100	64H	d
069	45H	E	101	65H	e
070	46H	F	102	66H	f
071	47H	G	103	67H	g
072	48H	H	104	68H	h
073	49H	I	105	69H	i
074	4AH	J	106	6AH	j
075	4BH	K	107	6BH	k
076	4CH	L	108	6CH	l
077	4DH	M	109	6DH	m
078	4EH	N	110	6EH	n
079	4FH	O	111	6FH	o
080	50H	P	112	70H	p
081	51H	Q	113	71H	q
082	52H	R	114	72H	r
083	53H	S	115	73H	s
084	54H	T	116	74H	t
085	55H	U	117	75H	u
086	56H	V	118	76H	v
087	57H	W	119	77H	w
088	58H	X	120	78H	x
089	59H	Y	121	79H	y
090	5AH	Z	122	7AH	z
091	5BH	[123	7BH	{
092	5CH	\	124	7CH	
093	5DH]	125	7DH	}
094	5EH	^	126	7EH	~
095	5FH	_	127	7FH	DEL

Dec=Decimal, Hex=Hexadecimal (H), CHR=Character,
 LF=Line Feed, FF=Form Feed, CR=Carriage Return,
 DEL=Rub Out

APPENDIX C
ERROR MESSAGES

C.1	Compile Time Error Messages	187
C.2	Runtime Error Messages	195
C.2.1	File System Errors	195
C.2.2	Other Runtime Errors	200
C.2.2.1	Memory Errors	200
C.2.2.2	Type REAL Arithmetic Errors	200
C.2.2.3	Type INTEGER*4 Arithmetic Errors	202
C.2.2.4	Other Errors	202

(

(

(

C.1 Compile Time Error Messages

Code	Message
1	Fatal error reading source
2	Non-numeric characters in label field
3	Too many continuation lines
4	Fatal end of file encountered
5	Label in continuation line
6	Missing field in metacommand
7	Cannot open file
8	Unrecognizable metacommand
9	Input file invalid format
10	Too many nested include files
11	Integer constant error
12	Real constant error
13	Too many digits in constant
14	Identifier too long
15	Character constant not closed
16	Zero length character constant
17	Invalid character in input
18	Integer constant expected
19	Label expected
20	Label error
21	Type expected
22	Integer constant expected
23	Extra characters at end of statement
24	"(" expected

- 25 Letter already used in IMPLICIT
- 26 ")" expected
- 27 Letter expected
- 28 Identifier expected
- 29 Dimensions expected
- 30 Array already dimensioned
- 31 Too many dimensions
- 32 Incompatible arguments
- 33 Identifier already has type
- 34 Identifier already declared
- 35 INTRINSIC FUNCTION not allowed here
- 36 Identifier must be a variable
- 37 Identifier must be a variable or the
current FUNCTION
- 38 "/" expected
- 39 Named COMMON block already saved
- 40 Variable already appears in COMMON
- 41 Variables in two different COMMON blocks
- 42 Number of subscripts conflicts with
declaration
- 43 Subscript out of range
- 44 Forces location conflict for items in COMMON
- 45 Forces location in negative direction
- 46 Forces location conflict
- 47 Statement number expected
- 48 CHARACTER and numeric items in same COMMON
block
- 49 CHARACTER and non character item conflict

- 50 Invalid symbol in expression
- 51 SUBROUTINE name in expression
- 52 INTEGER or REAL expected
- 53 INTEGER, REAL or CHARACTER expected
- 54 Types not compatible
- 55 LOGICAL expression expected
- 56 Too many subscripts
- 57 Too few subscripts
- 58 Variable expected
- 59 "=" expected
- 60 Size of CHARACTER items must agree
- 61 Assignment types do not match
- 62 SUBROUTINE name expected
- 63 Dummy argument not allowed
- 64 Dummy argument not allowed
- 65 Assumed size declarations only for dummy arrays
- 66 Adjustable size array declarations only for dummy arrays
- 67 Assumed size must be last dimension
- 68 Adjustable bound must be parameter or in COMMON
- 69 Adjustable bound must be simple integer variable
- 70 More than one main program
- 71 Size of named COMMON must agree
- 72 Dummy arguments not allowed
- 73 COMMON variables not allowed

74	SUBROUTINE, FUNCTION, or INTRINSIC names not allowed	
75	Subscript out of range	
76	Repeat count must be ≥ 1	(
77	Constant expected	
78	Type conflict	
79	Number of variables does not match	
80	Label not allowed	
81	No such INTRINSIC FUNCTION	
82	INTRINSIC FUNCTION type conflict	
83	Letter expected	
84	FUNCTION type conflict with previous call	
85	SUBROUTINE / FUNCTION already defined	
87	Argument type conflict	
88	SUBROUTINE / FUNCTION conflict with previous use	(
89	Unrecognizable statement	
90	CHARACTER FUNCTION not allowed	
91	Missing END statement	
93	Fewer actual than dummy arguments in call	
94	More actual than dummy arguments in call	
95	Argument type conflict	
96	SUBROUTINE / FUNCTION not defined	
98	CHARACTER size invalid ?	
100	Statement order	
101	Unrecognizable statement	(
102	Jump into block not allowed	
103	Label already used for FORMAT	

104 Label already defined
105 Jump to FORMAT not allowed
106 DO statement not allowed here
107 DO label must follow DO statement
108 ENDIF not allowed here
109 Matching IF missing
110 Improperly nested DO block in IF block
111 ELSEIF not allowed here
112 Matching IF missing
113 Improperly nested DO or ELSE block
114 "(" expected
115 ")" expected
116 THEN expected
117 Logical expression expected
118 ELSE not allowed here
119 Matching IF missing
120 GOTO not allowed here
121 GOTO not allowed here
122 Block IF not allowed here
123 Logical IF not allowed here
124 Arithmetic IF not allowed here
125 "," expected
126 Expression of wrong type
127 RETURN not allowed here
128 STOP not allowed here
129 END not allowed here
131 Label not defined

132	DO or IF block not terminated	
133	FORMAT not allowed here	
134	FORMAT label already referenced	
135	FORMAT label missing	(
136	Identifier expected	
137	Integer variable expected	
138	TO expected	
139	Integer expression expected	
140	ASSIGN statement missing	
141	Unrecognizable character constant	
142	Character constant expected	
143	Integer expression expected	
144	STATUS option expected	
145	Only character expression allowed	(
146	Conflicting options	
147	Option already defined	
148	Integer expression expected	
149	Unrecognizable option	
150	RECL= missing	
151	Adjustable arrays not allowed here	
152	End of statement encountered in implied DO	
153	Variable required as control for implied DO	
154	Expressions not allowed in I/O list	
155	REC= option already defined	
156	Integer expression expected	(
157	END= not allowed here	
158	END= already defined	

159 Unrecognizable I/O unit
160 Unrecognizable format in I/O
161 Options expected after ", "
162 Unrecognizable I/O list element
163 FORMAT not found
164 ASSIGN missing
165 Label used as FORMAT
166 Integer variable expected
167 Label defined more than once as format
203 CHARACTER FUNCTION not allowed
406 Unit zero must be formatted and sequential
407 ERR= already defined
408 Too many labels
409 Invalid size for this type
410 PRECISION expected
411 Integer type conflict
415 Dimension too big
420 Invalid FUNCTION call
421 INTRINSIC not allowed
501 Unrecognizable character
502 Blank not allowed in metacommand
503 Metacommand not allowed here
504 Size already defined
601 Out of range
701 CHARACTER type expected
703 Internal error
705 Internal error

706	Internal error
708	Internal error
709	CHARACTER type not expected
710	Internal error
711	Internal error
712	Internal error
713	Long integer conversion error
714	Cannot convert to single
715	Cannot convert to double
717	Internal error
802	Invalid radix
803	Starting location is odd
804	Real constant overflow
805	Integer constant too big
806	Missing actual argument
807	Variable too big
808	Data size exceeds max
809	Numeric expected
810	Numeric or CHARACTER expected

C.2 Runtime Error Messages

Runtime errors fall into two classes:

1. file system errors
2. nonfile system errors

Nonfile system errors include the following:

1. memory errors
2. type REAL arithmetic errors
3. type INTEGER*4 arithmetic errors
4. other errors

If you see an error message that is not listed, check your operating system manual, as the error may be an operating system error.

C.2.1 File System Errors

Code numbers 1000 through 1099 are status codes, always issued in conjunction with an OS status code.

Code	Message
1000	Write error when writing end of file
1002	Filename extension with more than 3 characters
1003	Error during creation of new file (Disk or directory full)
1004	Error during open of existing file (File not found)
1005	Filename with more than 8 or zero characters
1007	Total filename length over 21 characters
1008	Write error when advancing to next record
1009	File too big (Over 65535 logical sectors)
1010	Write error when seeking to direct record device

1011	Attempt to open a random file to a non-disk device
1012	Forward space or back space on a non-disk device
1013	Disk or directory full error during forward space or back space
1200	Format missing final ")"
1201	Sign not expected in input
1202	Sign not followed by digit in input
1203	Digit expected in input
1204	Missing N or Z after B in format
1205	Unexpected character in format
1206	Zero repetition factor in format not allowed
1207	Integer expected for w field in format
1208	Positive integer required for w field in format
1209	"." expected in format
1210	Integer expected for d field in format
1211	Integer expected for e field in format
1212	Positive integer required for e field in format
1213	Positive integer required for w field in format
1214	Hollerith field in format must not appear for reading
1215	Hollerith field in format requires repetition factor
1216	X field in format requires repetition factor
1217	P field in format requires repetition factor
1218	Integer appears before + or - in format
1219	Integer expected after + or - in format

- 1220 P format expected after signed repetition factor in format
- 1221 Maximum nesting level for formats exceeded
- (1222 ")" has repetition factor in format
- 1223 Integer followed by , illegal in format
- 1224 "." is illegal format control character
- 1225 Character constant must not appear in format for reading
- 1226 Character constant in format must not be repeated
- 1227 "/" in format must not be repeated
- 1228 "\" in format must not be repeated
- 1229 BN or BZ format control must not be repeated
- 1230 Attempt to reference unknown unit number
- (1231 Formatted I/O attempted on file opened as unformatted
- 1232 Format fails to begin with "("
- 1233 I format expected for integer read
- 1234 F or E format expected for real read
- 1235 Two "." characters in formatted real read
- 1236 Digit expected in formatted real read
- 1237 L format expected for logical read
- 1238 Blank logical field
- 1239 T or F expected in logical read
- 1240 A format expected for character read
- (1241 I format expected for integer write
- 1242 w field in F format not greater than d field + 1

- 1243 Scale factor out of range of d field in
 E format
- 1244 E or F format expected for real write
- 1245 L format expected for logical write (
- 1246 A format expected for character write
- 1247 Attempt to do unformatted I/O to a unit
 opened as formatted
- 1251 Integer overflow on input
- 1252 Too many bytes read from input record
- 1253 Too many bytes written to direct access
 unit record
- 1255 Attempt to do external I/O on a unit
 beyond end of file record
- 1256 Attempt to position a unit for direct
 access on a non-positive record number
- 1257 Attempt to do direct access to a unit
 opened as sequential (
- 1258 Unable to seek to file position
- 1260 Attempt to backspace unit connected to
 unblocked device
- 1261 Premature end of file of unformatted sequential
 file
- 1262 Invalid blocking in unformatted sequential file
- 1263 Incorrect physical record structure in
 unformatted file
- 1264 Attempt to do unformatted I/O to internal unit
- 1265 Attempt to put more than one record into
 internal unit
- 1266 Attempt to write more characters to
 internal unit than its length (
- 1267 EOF called on unknown unit
- 1268 Dynamic file allocation limit exceeded

1269 Scratch file opened for read

1270 Console I/O error

1272 File operation attempted after error
 encountered on previous operation

1273 Keyboard buffer overflow: too many bytes
 written to keyboard input record
 (Must be less than 132)

1274 Reading long integer

1275 Writing long integer

1281 Repeat field not on integer

1282 Multiple repeat character

1283 Invalid numeric data in list directed input

1284 List directed numeric items bigger than record
 size

1285 Invalid string in list directed input

1298 End of file encountered

1299 Integer variable not ASSIGNED a label used in
 assigned GOTO

C.2.2 Other Runtime Errors

Nonfile system error codes range from 2000 to 2999. In some cases, metacommands determine if errors are checked; in other cases, error codes are always checked.

C.2.2.1 Memory Errors

The heap is the storage area where MS-FORTRAN dynamically allocates storage for file control blocks. Since the stack and the heap grow toward each other, memory errors are all related; for example, a stack overflow can cause a "Heap is Invalid" error.

Code	Message
2000	Stack Overflow
2002	Heap is Invalid
2052	Signed Divide By Zero (This error appears only when \$DEBUG is set.)
2054	Signed Math Overflow
2084	INTEGER Zero to Negative Power

C.2.2.2 Type REAL Arithmetic Errors

Code	Message
2100	REAL Divide By Zero
2101	REAL Math Overflow
2102	SIN or COS Argument Range
2103	EXP Argument Range
2104	SQRT of Negative Argument
2105	LN of Non-Positive Argument
2106	TRUNC/ROUND Argument Range
2131	Tangent Argument Too Small
2132	Arcsine or Arccosine of REAL > 1.0

- 2133 Negative REAL to REAL Power
- 2134 REAL Zero to Negative Power
- 2135 REAL Math Underflow
- 2136 REAL Indefinite (uninitialized or previous error)
- 2137 Missing Arithmetic Processor
(You have linked your program with the runtime library intended for use with the 8087 numeric coprocessor, but there is no coprocessor on your system. Relink your program with the runtime library that emulates floating point arithmetic.)
- 2138 REAL IEEE Denormal Detected
(A very small real number was generated and may no longer be valid due to loss of significance.)
- 2139 REAL Precision Loss
(An arithmetic operation on the 8087 numeric coprocessor has generated a loss of numeric precision in the result of an operation.)
- 2140 REAL Arithmetic Processor Instruction Illegal Or Not Emulated
(An attempt was made to execute an illegal arithmetic coprocessor instruction, or the floating point emulator cannot emulate a legal coprocessor instruction.)

C.2.2.3 Type INTEGER*4 Arithmetic Errors

Code	Message
------	---------

2200	Long integer divided by zero
2201	Long integer math overflow
2234	Long integer zero to negative power

C.2.2.4 Other Errors

Code	Message
------	---------

2451	Assigned GOTO label not in list (This error appears only when \$DEBUG is set.)
------	-----------------------------------------------------------------------------------

INDEX

\$DEBUG, 163
\$DO66, 164
\$INCLUDE, 165
\$LINESIZE, 166
\$LIST, 167
\$NOLIST, 167
\$NOTSTRICT, 171
\$PAGE, 168
\$PAGESIZE, 169
\$STORAGE, 170
\$STRICT, 171
\$SUBTITLE, 172
\$TITLE, 173

* files, 120

.AND., 36
.EQ., 35
.GT., 35
.LE., 35
.LT., 35
.NE., 35
.NOT., 36
.OR., 36

ABS, 150
Absolute value functions,
150
ACOS, 152
Actual argument, 152

Adjustable-size array,
63, 64
AINT, 150
ALOG, 152
ALOG10, 152
Alphanumeric characters,
17
AMAX0, 151
AMAX1, 151
AMIN0, 151
AMIN1, 151
AMODS, 150
ANINT, 150
Apostrophe editing, 131
Arc cosine functions, 152
Arc sine functions, 152
Arc tangent functions,
152
Arguments, 155
Arithmetic
 assignments, 50
 errors, 200-202
 expressions, 32
 IF statement, 86
 integer operations, 34
 operators, 32
 type conversion, 51
Array
 adjustable-size,
 63, 64
 assumed-size, 64
 declarator, 63
 dimensions, 63
 element reference, 38
 maximum size, 63
 order of elements, 64

Index

Array (continued)
 subscript expression,
 38
ASIN, 152
ASSIGN statement, 49
Assigned GOTO statement,
 83
Assignment statement, 50
Assumed-size array, 64
ATAN, 152
ATAN2, 152

Backslash editing, 132
BACKSPACE statement, 53
Binary files, 182
Blank
 character, signifi-
 cance, 17
 interpretation, 133
Block IF statement, 88
BN, 133
BZ, 133

CALL statement, 54
Carriage control, 128
CHAR, 150
Character
 alphanumeric, 17
 blanks, 17
 constant, 28
 data type, 28
 editing, 138
 expressions, 34
 standard set, 13, 17
 TAB, 18
 variable, 28
Choosing largest value
 function, 151
Choosing smallest value
 function, 151
CLOSE statement, 57
Columns, 18
Comment lines, 19
Common block
 saving, 102
 size, 58
Common logarithm func-
 tions, 152

COMMON statement, 58
Compile time errors, 187
Compiler control, 3
Computational assignment
 statement, 50
Computed GOTO statement, (84
Constants
 double precision, 26
 integer, 24
 REAL*4, 25
 REAL*8, 26
 single precision, 25
Continuation lines, 19
CONTINUE statement, 60
COS, 152
COSH, 153
Cosine functions, 152

DABS, 150
DACOS, 152
DASIN, 152
DATA, 45, 61
Data types, 11
 basic, listed, 22 (26
 double precision, 26
 ranks, 34
 REAL*4, 25
 REAL*8, 26
 character, 28
 integer, 24
 logical, 27
 memory requirements,
 23
DATAN, 152
DATAN2, 152
DBLE, 150
DCOS, 152
DCOSH, 153
DDIM, 151
DEBUG metacommand, 163
Defining statements, 20
DEXP, 151
DIM, 151
Dimension declarators, 63
DIMENSION statement, 63 (63
DINT, 150
Direct
 access files, 118, 122
 devices, files, 123

DLOG, 152
 DLOG10, 152
 DMAX1, 151
 DMIN1, 151
 DMOD, 150
 DNINT, 150
 DO statement, 65
 DO66 metacommand, 164
 Double precision
 constant, 26
 data type, 26
 exponent, 27
 range, 26
 DPROD, 151
 DSIGN, 151
 DSIN, 152
 DSINH, 153
 DSQRT, 151
 DTAN, 152
 DTANH, 153

 Edit descriptors
 A, 138
 BN, 133
 BZ, 133
 D, 137
 E, 135
 Fw.d, 134
 Iw, 134
 L<w>, 138
 <k>P, 132
 \, 132
 nonrepeatable, 79, 80
 numeric, 134
 repeatable, 79
 types, 131
 Editing
 apostrophe, 131
 backslash, 132
 character, 138
 Hollerith, 131
 integer, 134
 logical, 138
 numeric, 134
 positional, 131
 real, 134
 short records, 133
 slash, 132
 Elements of I/O state-
 ments, 124

ELSE statement, 68
 ELSEIF statement, 69
 description, 69
 in block IF statement,
 89
 END statement, 71
 End-of-file function,
 124, 153
 END=
 option, 117
 specifier, 181
 ENDFILE statement, 72
 ENDIF statement, 73
 EOF function, 124, 153
 EQUIVALENCE statement, 74
 ERR= specifier, 180
 Error messages, 185
 Errors
 compile time, 187
 file system, 195
 memory, 200
 runtime, 195, 200
 type INTEGER*4 arith-
 metic, 202
 type REAL arithmetic,
 200
 Evaluation rules, 38
 EXP, 151
 Explicitly opened files,
 120
 Exponent
 double precision, 27
 single precision, 26
 Exponential functions,
 152
 Expressions, 9, 31
 arithmetic, 32, 38
 character, 34
 logical, 36
 relational, 35
 restrictions, 38
 External
 files, 7
 unit specifier, 119
 EXTERNAL statement, 78

 File system errors, 195
 FILE= specifier, 180
 Files,
 access methods, 118

Index

Files (continued)

- binary, 182
- commonly used structures, 120
- device association, 123
- direct access, 117, 118, 122
- explicitly opened, 120
- external, 116
- formatted, 117
- internal, 116, 118, 123
- keyboard and screen (* files), 120
- limitations, 120
- names, 116
- NEW, 123
- OLD, 123
- position, 116
- properties, 7, 116, 118
- sequential access, 117, 118
- special properties, 118
- structure, 120
- system, 116
- unformatted, 117

FLOAT, 150

Formal argument, 156

Format

- controller, 130
- specification, 128

FORMAT statement, 79, 128

Formatted

- files, 121
- records, 115

FORTRAN

- character set, 17
- file system, 115
- I/O system, 6
- identifiers, 10, 29
- learning about, 10
- main program, 4, 21, 147
- names, 10, 29
- program units, 4, 20, 147
- statements, 8, 19, 39
- subprogram, 21
- subroutines, 4, 147

Function

- absolute value, 150
- arc cosine, 152
- arc sine, 152
- arc tangent, 152
- arguments, 155
- choosing largest value, 151
- choosing smallest value, 151
- common logarithm, 152
- cosine, 152
- description of, 4, 148
- EOF (end-of-file), 124, 153
- exponential, 151
- external, 148
- hyperbolic cosine, 153
- hyperbolic sine, 153
- hyperbolic tangent, 153
- intrinsic, 93, 149-154
- lexically greater than, 154
- lexically greater than or equal, 153
- lexically less than or equal, 153
- lexically less than, 153
- name, 81
- natural logarithm, 152
- nearest integer, 150
- nearest whole number, 150
- positive difference, 151
- REAL*8 product, 151
- recursive call, 82
- reference to, 148
- remaindering, 150
- sign, 151
- sine, 152
- square root, 151
- tangent, 152
- transfer of sign, 151
- truncation, 150
- type conversion, 150
- types of, 148

FUNCTION statement, 81

- Global name, 30
- GOTO statement, 83, 84, 85
- Hollerith editing, 131
- Hyperbolic
 - cosine functions, 153
 - sine functions, 153
 - tangent functions, 153
- I/O, See Input/Output
- IABS, 150
- ICHAR, 150
- IDIM, 151
- IDINT, 150
- IDNINT, 150
- IF-levels, 89
- IFIX, 150
- IMPLICIT statement, 91
- Implied DO lists, 127
- INCLUDE metacommand, 165
- Initial lines, 18
- Input entity, 126
- Input/Output
 - BACKSPACE statement, 53
 - carriage control, 128
 - character expression, 94
 - CLOSE statement, 57
 - ENDFILE statement, 72
 - entities, 126
 - format specifier, 125
 - FORMAT statement, 79
 - iolist, 125, 129
 - OPEN statement, 94
 - READ statement, 98
 - REWIND statement, 101
 - statements, 6, 8, 48, 124, 125
 - unit specifier, 125
 - WRITE statement, 110
- INT, 150
- Integer
 - constants, 24
 - data types, 24
 - division, 33
 - editing, 134
 - expression, 125
 - variable name, 83
- INTEGER*2, 24, 149-154
- INTEGER*4, 24, 149-154
- Internal
 - files, 7, 116, 118, 122
 - unit specifier, 119
- Intrinsic function
 - name, 93
 - list of, 149-154
- INTRINSIC statement, 93
- Introduction to
 - MS-FORTRAN, vii
- iolist, 126
- ISIGN, 151
- Label assignment statement, 49
- Language overview, 1
- Learning about FORTRAN, x
- Lexically greater than
 - function, 153
- Lexically greater than or equal function, 153
- Lexically less than
 - function, 153
- Lexically less than or equal function, 153
- LGE, 153
- LGT, 153
- Limitations, 123
- Lines, 12, 18
- LINESIZE metacommand, 166
- LIST metacommand, 167
- List-directed
 - input, 139
 - output, 142
- LLE, 153
- LLT, 153
- Local name, 30
- Logical
 - .FALSE, 27
 - .TRUE, 27
 - data type, 27
 - expressions, 36
 - IF, 87
 - operators, 36

Index

Main program, 147
MAX0, 151
MAX1, 151
Memory errors, 200
Metacommands
 available, 3
 \$DEBUG, 163
 \$DO66, 164
 \$INCLUDE, 165
 \$LINE SIZE, 166
 \$LIST, 167
 \$NOLIST, 167
 \$NOTSTRICT, 171
 \$PAGE, 168
 \$PAGE SIZE, 169
 \$STORAGE, 170
 \$STRICT, 171
 \$SUBTITLE, 172
 \$TITLE, 173
MIN0, 151
MIN1, 151
MODS, 150

Name

 common block, 58
 common data block, 30
 external subroutine,
 78
 formal argument, 106
 global, 30
 integer variable, 83
 local, 30
 program, 97
 restrictions, 10
 subroutine, 54, 106
 symbolic, 109
 undeclared, 31
 user-defined, 108
Natural logarithm
 functions, 152
Nearest integer
 functions, 150
Nearest whole number
 functions, 150
NEW files, 123
NINT, 150
NOLIST metacommand, 167
Nonfile system errors,
 200
Nonrepeatable edit de-
 scriptors, 131

Normal termination, 57
Notation, 17
 FORTRAN, 17
 statement syntax, ix
NOTSTRICT metacommand,
 171
Numeric editing, 134

OLD files, 123
OPEN statement, 94
Operators
 arithmetic, 32
 classes, 37
 logical, 36
 precedence, 37
 relational, 35
Output
 see also Input/Output
 entities, 126
Overview of MS-FORTRAN, 1

PAGE metacommand, 168
PAGE SIZE metacommand, 169
PAUSE statement, 96
Positional editing, 80,
 131
Positive difference
 functions, 151
Precedence of operators,
 37
PROGRAM statement, 97

Radix specifier, 24
READ statement, 98
Reading short records,
 133
Real
 constant, 26
 editing, 134, 135, 137
REAL*4, 25, 149-154
REAL*8, 26, 149-154
REC= option, 118
Records
 endfile, 115
 formatted, 115
 properties, 7

Records (continued)
 short, 133
 unformatted, 115
 Recursive functions
 calls, 82
 Reference manual organi-
 zation, viii
 Relational
 expressions, 35
 operators, 35
 Remaindering functions,
 150
 Repeat
 factor, 61
 specification, 79
 Repeatable edit descrip-
 tors, 134
 RETURN statement, 100
 REWIND statement, 101
 Runtime errors
 arithmetic, 200-202
 classes of, 195
 file system, 195
 memory errors, 200
 nonfile system, 200
 other, 202

 SAVE statement, 102
 Scale factor editing, 80,
 132, 136
 Sequential properties,
 118
 Short records, 133
 SIGN, 151
 SIN, 152
 Sine functions, 152
 Single precision data
 type, 25
 SINH, 153
 Slash editing, 80, 132
 SNGL, 150
 Specification statement,
 44, 45
 Specifiers on I/O state-
 ments, 180
 SQRT, 151
 Square root functions,
 151

Statement
 arithmetic IF, 86
 ASSIGN, 49
 assigned GOTO, 83
 assignment, 50
 BACKSPACE, 53
 block IF, 88
 CALL, 54
 categories, 8, 43
 CLOSE, 57
 COMMON, 58
 computed GOTO, 84
 CONTINUE, 60
 DATA, 45, 61
 definition, 20
 DIMENSION, 63
 directory, 49
 DO, 65
 ELSE, 68
 ELSEIF, 69
 END, 71
 ENDFILE, 72
 ENDIF, 73
 EQUIVALENCE, 74
 EXTERNAL, 78
 FORMAT, 79
 FUNCTION, 81, 148
 functions, 155
 GOTO, 83, 84, 85
 IF, 86, 87
 IF THEN ELSE, 88
 I/O, elements, 125
 IMPLICIT, 91
 INTRINSIC, 93, 149-154
 labels, 20, 49, 86
 lines, 18, 19
 logical IF, 87
 nesting rules, 89
 OPEN, 94
 ordering, 20-22
 PAUSE, 96
 PROGRAM, 97, 147
 READ, 98
 RETURN, 100
 REWIND, 101
 SAVE, 102
 specification, 45
 statement function,
 103
 STOP, 105

Index

- Statement (continued)
 - SUBROUTINE, 106, 147
 - syntax notation, ix
 - type, 108
 - unconditional GOTO, 85
 - WRITE, 110
- Statement function statement, 103
- STATUS= specifier, 180
- STORAGE metacommand, 170
- STRICT metacommand, 171
- SUBROUTINE statement, 106
- Subroutines, 4, 147
- Subscript expression, 38
- SUBTITLE metacommand, 172
- Symbolic name, 109
- Syntax notation, ix
-
- TAB character, 18
- TAN, 152
- Tangent functions, 152
- TANH, 153
- Terms and concepts, 15
- TITLE metacommand, 173
- Transfer of sign functions, 151
-
- Truncation functions, 150
- Type, data, 11
- Type conversion
 - arithmetic operands, 33
 - functions, 149-154
 - real values, 52
- Type INTEGER*4 arithmetic errors, 202
- Type REAL arithmetic errors, 200
- Type statement, 108
-
- Unconditional GOTO statement, 85
- Undeclared FORTRAN names, 31
- Unformatted files, 117, 122
- Units, 119
- User-defined names, 108
-
- WRITE statement, 110

Name _____
Street _____
City _____ State _____ Zip _____
Phone _____ Date _____

Instructions

Use this form to report software bugs, documentation errors, or suggested enhancements. Mail the form to Microsoft.

Category

_____ Software Problem _____ Documentation Problem
_____ Software Enhancement (Document # _____)
_____ Other

Software Description

Microsoft Product _____
Rev. _____ Registration # _____
Operating System _____
Rev. _____ Supplier _____
Other Software Used _____
Rev. _____ Supplier _____

Hardware Description

Manufacturer _____ CPU _____ Memory _____ KB
Disk Size _____" Density: _____ Sides: _____
Single _____ Single _____
Double _____ Double _____
Peripherals _____

Problem Description

Describe the problem. (Also describe how to reproduce it, and your diagnosis and suggested correction.) Attach a listing if available.

Microsoft Use Only

Tech Support _____

Date Received _____

Routing Code _____

Date Resolved _____

Report Number _____

Action Taken:



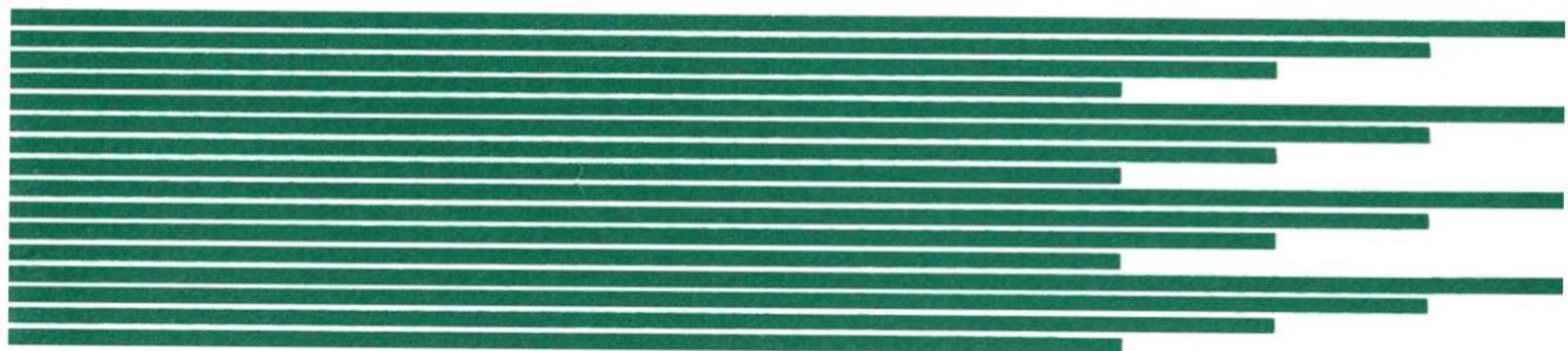
Customer
Service
Plan

Service
Support
Warranty

IMPORTANT!

Send in your product
registration card

TODAY!



For Software and Hardware Products

Product Registration

To take advantage of Microsoft's customer services, you must first register your product with Microsoft. The Microsoft product registration card is attached to this booklet. We encourage you to complete and mail the card promptly.

Registration of your product gives you access to:

1. Microsoft Product Replacement Plan that allows you to replace defective products, even after the warranty expires, for a small charge.
2. Microsoft Product Upgrade Plan that allows you to buy new versions of certain software products as they are released, at a fraction of the cost of the original product.

Details of both of these services are described in the following sections.

In addition, registering your product makes it simpler for you to obtain warranty service. The registration card provides proof of purchase information, so no receipts or other records need be supplied with a warranty claim.

Product Upgrade Plan

Microsoft periodically releases new versions of certain software products, incorporating new or improved features. The Microsoft Product Upgrade Plan enables you to purchase an enhanced version of your Microsoft product at a fraction of its regular price.

When a new version of a product is released, registered product owners are notified via a product upgrade notice. The notice details the enhancements made in the new version, the upgrade price, and ordering information.

IMPORTANT: To qualify for the Product Upgrade Plan, you must be a registered product owner. To register your Microsoft product, complete the product registration card attached to this booklet and return it to Microsoft.

Disclaimer of Liability for Use and the Results of Use

The Microsoft programs are licensed solely on an "as is" basis. The entire risk as to its quality and performance is assumed by the purchaser. MICROSOFT CORPORATION DOES NOT GUARANTEE, WARRANT, NOR MAKE ANY REPRESENTATION REGARDING THE USE OF, OR THE RESULTS OF THE USE OF, THE PROGRAMS IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS, OR OTHERWISE; AND THE PURCHASER RELIES ON THE PROGRAMS AND THE RESULTS SOLELY AT HIS OWN RISK. Microsoft Corporation assumes no liability for any direct, indirect, incidental or consequential, special or exemplary damages, regardless of its having been advised of the possibility of such damages. A full description of the terms of the disclaimer is in the License Agreement.

Microsoft Software Limited Warranty

The diskette on which your Microsoft program is recorded is warranted to be free of defects in materials and workmanship under normal use for a period of 90 days from date of product purchase.

This limited warranty applies to the original purchaser only and to the recording medium (diskette) only, not the information encoded on it. This warranty covers diskettes included in Microsoft hardware/software packages, such as the Microsoft® SoftCard™ system products and the Microsoft® RAMCard™ memory board for the IBM® PC.

A full explanation of the limited warranty terms is included in this booklet in the License Agreement section.

Software Replacement Plan

Microsoft will replace a product diskette free of charge if it proves defective during the warranty period. (See the License Agreement in this booklet for more information on the warranty period.) To receive warranty replacement, you must provide proof of purchase or have registered your product with Microsoft.

After the limited warranty has expired and if your product registration card is on file at Microsoft, we will replace the defective diskette for a nominal cost. You must return the product registration card to take advantage of this service.

If you have a defective diskette, follow these procedures to obtain a replacement:

1. Obtain a Return Authorization (RA) number from Microsoft.
2. Complete the product replacement card attached to this booklet and request either warranty or out-of-warranty replacement.
3. Mail the replacement card along with the defective diskette to Microsoft's Customer Service Department. The address is listed on the replacement card.
4. Include your payment if the limited warranty has expired. Check the Replacement Price List included with this booklet for the replacement cost of a defective diskette. The price listed is *per diskette*.

To obtain an RA number, call the Customer Service Department at Microsoft. Be ready to furnish the following information:

1. What is the product?
2. Reason you are returning it?

Microsoft Hardware Limited Warranty

Microsoft hardware components only include circuit cards and the mechanical mouse.

If a hardware component is included with your Microsoft product, the component is warranted to be free of defects in materials and workmanship under normal use for a period of one year from date of product purchase.

This limited warranty applies to the original product purchaser only and to the hardware component only, not the application for which it is used.

A full description of hardware limited warranty terms is included in the License Agreement section of this booklet.

Hardware Replacement Plan

Microsoft will replace a hardware component free of charge if it proves defective during the warranty period. To receive warranty replacement, you must provide proof of purchase or have registered your product with Microsoft.

After the limited warranty has expired and if your product registration card is on file at Microsoft, we will replace a defective hardware component for the cost set forth on the Replacement Price List included with this booklet. You must return the product registration card to take advantage of this service.

If you have a defective hardware component, follow these procedures to obtain a replacement:

1. Obtain a Return Authorization (RA) number from Microsoft.
2. Complete the product replacement card attached to this booklet and request either warranty or out-of-warranty replacement.
3. Mail the replacement card along with the defective hardware component to Microsoft's Customer Service Department. The address is listed on the replacement card.
4. Include your payment if the warranty has expired. Check the Replacement Price List included with this booklet for the replacement cost of a defective hardware component. The price listed is *per hardware component*.

To obtain an RA number, call the Customer Service Department at Microsoft, (206) 828-8080. Be ready to furnish the following information:

1. What is the product?
2. Reason you are returning it?
3. Your name and address.

Microsoft License Agreement

CAREFULLY READ ALL THE TERMS AND CONDITIONS OF THIS AGREEMENT PRIOR TO OPENING THIS PACKAGE. OPENING THIS PACKAGE INDICATES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS.

If you do not agree to these terms and conditions, return the unopened diskette package and the other components of this product to the place of purchase and your money will be refunded. No refunds will be given for products which have opened diskette packages or missing components.

1. LICENSE: You have the non-exclusive right to use the enclosed program. This program can only be used on a single computer. You may physically transfer the program from one computer to another provided that the program is used on only one computer at a time. You may not electronically transfer the program from one computer to another over a network. You may not distribute copies of the program or documentation to others. You may not modify or translate the program or related documentation without the prior written consent of Microsoft.

YOU MAY NOT USE, COPY, MODIFY, OR TRANSFER THE PROGRAM OR DOCUMENTATION, OR ANY COPY EXCEPT AS EXPRESSLY PROVIDED IN THIS AGREEMENT.

2. BACK-UP AND TRANSFER: You may make one (1) copy of the program solely for back-up purposes. You must reproduce and include the copyright notice on the back-up copy. You may transfer and license the product to another party if the other party agrees to the terms and conditions of this Agreement and completes and returns a Registration Card to Microsoft. If you transfer the program you must at the same time transfer the documentation and back-up copy or transfer the documentation and destroy the back-up copy.

3. COPYRIGHT: The program and its related documentation are copyrighted. You may not copy the program or its documentation except as for back-up purposes and to load the program into the computer as part of executing the program. All other copies of the program and its documentation are in violation of this Agreement.

4. TERM: This license is effective until terminated. You may terminate it by destroying the program and documentation and all copies thereof. This license will also terminate if you fail to comply with any term or condition of this Agreement. You agree upon such termination to destroy all copies of the program and documentation.

5. HARDWARE COMPONENTS: Microsoft product hardware components only include circuit cards and the mechanical mouse.

6. LIMITED WARRANTY: THE PROGRAM IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE PROGRAM IS ASSUMED BY YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU (AND NOT MICROSOFT OR ITS DEALERS) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. FURTHER, MICROSOFT DOES NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OF, OR THE RESULTS OF THE USE OF, THE PROGRAM IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS, OR OTHERWISE; AND YOU RELY ON THE PROGRAM AND RESULTS SOLELY AT YOUR OWN RISK.

Microsoft does warrant to the original licensee that the diskette(s) on which the program is recorded be free from defects in materials and workmanship under normal use and service for a period of ninety (90) days from the date of delivery as evidenced by a copy of your receipt. Microsoft warrants to the original licensee that the hardware components included in this package are free from defects in materials and workmanship for a period of one year from the date of delivery to you as evidenced by a copy of your receipt. Microsoft's entire liability and your exclusive remedy shall be replacement of the diskette or hardware component not meeting Microsoft's limited warranty and which is returned to Microsoft with a copy of your receipt. If failure of the diskette or hardware component has resulted from accident, abuse or misapplication of the product, then Microsoft shall have no responsibility to replace the diskette or hardware component under this Limited Warranty. In the event of replacement of the hardware component the replacement will be warranted for the remainder of the original one (1) year period or 30 days, whichever is longer.

THE ABOVE IS THE ONLY WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE THAT IS MADE BY MICROSOFT ON THIS MICROSOFT PRODUCT. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

NEITHER MICROSOFT NOR ANYONE ELSE WHO HAS BEEN INVOLVED IN THE CREATION, PRODUCTION, OR DELIVERY OF THIS PROGRAM SHALL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, OR INCIDENTAL DAMAGES ARISING OUT OF THE USE, THE RESULTS OF USE, OR INABILITY TO USE SUCH PRODUCT EVEN IF MICROSOFT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR CLAIM. SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

7. UPDATE POLICY: In order to be able to obtain updates of the program, the licensee and persons to whom the program is transferred in accordance with this Agreement must complete and return the attached Registration Card to Microsoft. IF THIS REGISTRATION CARD HAS NOT BEEN RECEIVED BY MICROSOFT, MICROSOFT IS UNDER NO OBLIGATION TO MAKE AVAILABLE TO YOU ANY UPDATES EVEN THOUGH YOU HAVE MADE PAYMENT OF THE APPLICABLE UPDATE FEE.

8. MISC.: This license agreement shall be governed by the laws of the State of Washington and shall inure to the benefit of Microsoft Corporation, its successors, administrators, heirs and assigns.

9. ACKNOWLEDGEMENT: YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT, AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU ALSO AGREE THAT THIS AGREEMENT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF AGREEMENT BETWEEN THE PARTIES AND SUPERCEDES ALL PROPOSALS OR PRIOR AGREEMENTS, VERBAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN THE PARTIES RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

Should you have any questions concerning this Agreement, please contact in writing Microsoft, Customer Sales and Service, 10700 Northup Way, Bellevue, WA 98004.

Microsoft is a registered trademark; SoftCard and RAMCard are trademarks of Microsoft Corporation.

Microsoft Product Replacement Price List

Product Components	Out-of-Warranty Replacement Price
Any Microsoft diskette	\$ 25.00 ea.
Apple II 16K RAMCard circuit card	\$ 45.00 ea.
IBM PC 64K RAMCard circuit card	\$140.00 ea.
RAMChips (9 chips per package)	
\$55.00 per package or	\$ 7.00 ea.
SystemCard 64K circuit card	\$160.00 ea.
SystemCArd 256K circuit card	\$320.00 ea.
SoftCard system circuit card	\$ 95.00 ea.
Premium SoftCard Ile system circuit card	\$125.00 ea.
Mouse — bus system	
Bus mouse	\$ 75.00 ea.
Mouse circuit card	\$ 35.00 ea.
Mouse — serial system	
Serial mouse	\$110.00 ea.

Prices subject to change without notice. Effective 7/1/83

Replacement Order Card

Please use this card when ordering a replacement for a defective Microsoft product. To validate a replacement request for a product under limited warranty, include proof of purchase or indicate that your product registration card is on file at Microsoft. To validate a request for out-of-warranty replacement, your product must be registered with Microsoft.

This replacement order card must be accompanied by the defective product. If the product limited warranty has expired, please include payment for the replacement.

Name_____

Company_____

Address_____

City_____

State_____ Zip_____ Country_____

Phone (____) _____ TELEX_____

Name of product as it appears on package_____

Date of purchase: Month_____ Day_____ Year_____

If warranty has expired, I enclose payment in the amount of \$_____

☐ Check or money order ☐ VISA ☐ MasterCard

Bank card number_____ Expiration date_____

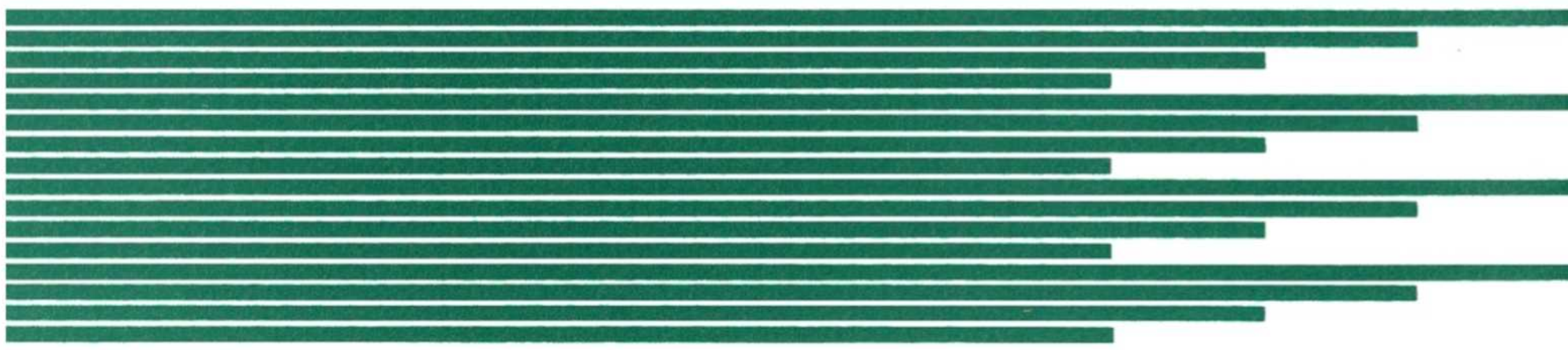
Authorized signature_____

Return Authorization Number*_____

Reason for return_____

Mail to: Customer Service Department
Microsoft Distribution Center
Parmac Business Park
11005 117th Place N.E.
Building C
Kirkland, WA 98033

*Your RA number must be written on the outside of the product package you are returning to Microsoft. See Software and Hardware Replacement Plan sections of this booklet for details on RA numbers.



MICROSOFT CORPORATION
10700 NORTHUP WAY
BELLEVUE, WASHINGTON 98004

Part No. CSP00